Dev manual for Toggle

Version 2.0

Authoring by Toggle dev Team

May 3, 2017

Contents

Chapter 1

Introduction

1.1 To whom this manual is addressed

The current manual is addressed to new TOGGLE developers, *i.e.* people wanting to implement new tools in the TOGGLE framework. If you just want to use already existing TOGGLE functions, you do not need to read it, you can go directly to the user manual on the website of the project.

1.2 General things about the TOGGLE github

Developers are required to work from the TOGGLE-dev github, accessible at `https://github.com/SouthGreenPlatform/TOGGLE-DEV`.

1.2.1 Preparing your working environment - Users registred on the TOGGLE-DEV github

You first have to clone the TOGGLE-DEV current version using the following commands:

```
#Cloning
git clone https://github.com/SouthGreenPlatform/TOGGLE-DEV /path/for/cloning

#Moving to the cloned folder
cd /path/for/cloning
```

Then, create your own development branch using the following commands:

```
#Create a branch
git branch branchName

#Switch to this branch
git checkout branchName

#Make a change then perform the fist commit
git commit -m "My comment" changedFile
```

```
#Push this local branch to GitHub
git push https://github.com/SouthGreenPlatform/TOGGLE-DEV.git branchName
```

This will prevent any regression in the current version and thus allow a reliable development.

Integration of new branchs will be perfomed by power users under request on the github. The integration depends on the correct application of the following recommandations, especially tests.

1.2.2 Preparing your working environment - Users NOT registred on the TOGGLE-DEV github

If you have no rights to work on the TOGGLE-DEV github, or you want to change it for your specific usage, please fork the repository using the GitHub forking manual.

1.3 General things about the conventions and nomenclatures in TOGGLE

In TOGGLE, the nomenclature is quite the same for all filenames, variables, modules or functions.
The way we will name a variable representing the output BAM file e.g. is **bamOutput**, thus all in lowercases, upper case being used to separate words.
A multiple words function such as the picard-tools CreateSequenceDictionary one will thus be **picardToolsCreateSequenceDictionary**.

Chapter 2

Creating a new module

A module is a set of functions related to each others, either because they came from the same software suite (`gatk.pm`,`bwa.pm`,...), or that they impact the same types of files (`fastqUtils.pm`).

2.1 Names

The name of the *Perl* module must be explicite. Do not use weird names such as "myTestModule.pm" to publish on the GitHub. Generally the name is related to the function target (software or format).

2.2 Requirements and Declaration

All modules created for TOGGLE must be structured as follows, with the same preamble:

```
1  package myName;
2
3  #
       ##############################################################################
4  #
5  # Copyright 2014-2017 IRD-CIRAD-INRA-ADNid-YOUR INSTITUTE
6  #
7  # This program is free software; you can redistribute it and/
       or modify
8  # it under the terms of the GNU General Public License as
       published by
9  # the Free Software Foundation; either version 3 of the
       License, or
10 # (at your option) any later version.
11 #
12 # This program is distributed in the hope that it will be
       useful,
13 # but WITHOUT ANY WARRANTY; without even the implied warranty
       of
14 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
```

```
15 # GNU General Public License for more details.
16 #
17 # You should have received a copy of the GNU General Public
     License
18 # along with this program; if not, see <http://www.gnu.org/
     licenses/> or
19 # write to the Free Software Foundation, Inc.,
20 # 51 Franklin Street, Fifth Floor, Boston,
21 # MA 02110-1301, USA.
22 #
23 # You should have received a copy of the CeCILL-C license with
      this program.
24 #If not see <http://www.cecill.info/licences/Licence_CeCILL-
     C_V1-en.txt>
25 #
26 # Intellectual property belongs to IRD, CIRAD and South Green
     developpement plateform for all versions, to ADNid for v2
     and latter versions, to INRA for v3 and latter versions
     and YOUR INSTITUTE for the current and latter versions
27 # Version 1 written by Cecile Monat, Ayite Kougbeadjo,
     Christine Tranchant, Cedric Farcy, Mawusse Agbessi,
     Maryline Summo, and Francois Sabot
28 # Version 2 written by Cecile Monat, Christine Tranchant,
     Cedric Farcy, Enrique Ortega-Abboud, Julie Orjuela-Bouniol
     , Sebastien Ravel, Souhila Amanzougarene, and Francois
     Sabot
29 # Version 3 written by Cecile Monat, Christine Tranchant,
     Laura Helou, Abdoulaye Diallo, Julie Orjuela-Bouniol,
     Sebastien Ravel, Gautier Sarah, and Francois Sabot
30 # Current version written by YOUR NAME and v3 authors
31 #
32 #
     ################################################################################

33
34 use strict;
35 use warnings;
36 use localConfig;
37 use toolbox;
38
39  sub foo{}
40
41  sub bar{}
42
43  1;
44
```

The licence must be conserved as given, except for an addition of the current developer name and institute.

The `use` lines are also mandatory to have access to the `toolbox` function (`run`,...), as described latter, as well as to the softwares location (`localConfig.pm` module).

Chapter 3

Creating a new function

Here is an example of a currently developed function

```
1  ##SAMTOOLS SORT
2  #Sort alignments by leftmost coordinates.
3  sub samToolsSort
4  {
5       my($bamFileIn,$bamFileOut,$optionsHachees)=@_;
6       if (toolbox::sizeFile($bamFileIn)==1)
7       { ##Check if entry file exist and is not empty
8
9            #Check if the format is correct
10           if (checkFormat::checkFormatSamOrBam($bamFileIn)==0)
11           {#The file is not a BAM/SAM file
12                toolbox::exportLog("ERROR: samTools::
     samToolsSort : The file $bamFileIn is not a SAM/BAM file\n
     ",0);
13                return 0;
14           }
15
16           my $options="";
17
18           if ($optionsHachees)
19           {
20                $options=toolbox::extractOptions(
     $optionsHachees);
21           }
22
23           #The current samtools sort version requires the -T
     option, ie temp prefix
24           my $tempPrefix = $bamFileOut;
25           $tempPrefix =~ s/\.bam/_temp/;
26
27           my $command=$samtools." sort ".$options." -o ".
     $bamFileOut." -T ".$tempPrefix." ".$bamFileIn;
28
29           #Execute command
30           if(toolbox::run($command)==1)
31           {
```

```
32                  return 1;#Command Ok
33              }
34              else
35              {
36                  toolbox::exportLog("ERROR: samTools::
    samToolsSort : Uncorrectly done\n",0);
37                  return 0;#Command not Ok
38              }
39          }
40          else
41          {
42              toolbox::exportLog("ERROR: samTools::samToolsSort :
    The file $bamFileIn is uncorrect\n",0);
43              return 0;#File not Ok
44          }
45  }
```

ALL THE FUNCTIONS MUST BE UNITARY, *i.e.* the shortest possible. All system calls must be performed throught the `toolbox::run` function

3.1 Nomenclature, Indentation and commentaries

As explained earlier, the names of variables and functions must be **functionName**.

Indentation is mandatory, as well as commentaries.

3.2 Basic structure of the function

A function will be designed as follows:

1. Picking up input data, output data (if any) and options
2. Verifying the input format, if any
3. Creating the output file name if not supplied already
4. Picking up the options in a text format (using `toolbox::extractOptions` function
5. Creating the command line
6. Sending command line to `toolbox::run` using a *if*
7. Sending log to `toolbox::exportLog` function to report errors

3.3 The `toolbox::exportLog` and `toolbox::run` functions

3.3.1 `toolbox::exportLog`

`toolbox::exportLog` is a intreasic feature in TOGGLE that will fill the various logs all along the pipeline running.

In a basic way, you can send any message to the current logs. **INFOS** and **WARNING** messages will not kill the current process, while **ERROR** will.

The numerical values at the end of the command arguments represent the state of the command and will send the text to a given log:

0 and 2 : ERROR and WARNING respectively, will send the text in the error log (log.e). Note that a WARNING (2) will not stop the running!

1 : INFOS, will send the text in the output log (log.o).

To construct a message, please follow the current nomenclature:

INFOS : toolbox::exportLog("INFOS: myModule::myFunction : Coffee is ready",1);

WARNINGS : toolbox::exportLog("WARNING: myModule::myFunction : Coffee is not ready yet",2);

ERRORS : toolbox::exportLog("ERROR: myModule::myFunction : No coffee left!!",0);

This function is highly complex, please do not modify it without the agreement of TOGGLE mainteners!

3.3.2 `toolbox::run`

This function will launch any command sent to it as argument (text or scalar) to the system, and will recover the exit status of the command. It will write the exact launched command in the output log, and any STDOUT also. All errors will be send to the error log and could drive to the stop of the pipeline.

To use this, respect the following nomenclature:

```
1 toolbox::run("my command to be launched");
```

As for the previous function, `toolbox::run` is an intreasic function that cannot be modified except by mainteners.

3.4 The code itself

Let's come back to our example:

```
1
2 ##SAMTOOLS SORT
3 #Sort alignments by leftmost coordinates.
4 sub samToolsSort
5 {
6       ...
7 }
```

The `sub` is preceded by commentaries about the function and what it does

```
1
2 ...
3       my($bamFileIn,$bamFileOut,$optionsHachees)=@_;
4
5 ...
```

input file and options are recovered through references.

```
1
2       if (toolbox::sizeFile($bamFileIn)==1)
3       { ##Check if entry file exist and is not empty
4
5           #Check if the format is correct
6           if (checkFormat::checkFormatSamOrBam($bamFileIn)==0)
7           {#The file is not a BAM/SAM file
8               toolbox::exportLog("ERROR: samTools::
    samToolsSort : The file $bamFileIn is not a SAM/BAM file\n
    ",0);
9               return 0;
10          }
11
12          MY CORE COMMAND
13      }
14      else
15      {
16          toolbox::exportLog("ERROR: samTools::samToolsSort :
    The file $bamFileIn is uncorrect\n",0);
17          return 0;#File not Ok
18      }
```

We check if the input file exists (`toolbox::sizeFile`) and if the file is a SAM or a BAM (`toolbox::checkSamOrBamFormat`). If any error appears (empty file, wrong format), the script is stopped and logs filled using the `toolbox::exportLog function`.

```
1           my $options="";
2
3           if ($optionsHachees)
4           {
5               $options=toolbox::extractOptions(
    $optionsHachees);
6           }
7
8           #The current samtools sort version requires the -T
    option, ie temp prefix
9           my $tempPrefix = $bamFileOut;
10          $tempPrefix =~ s/\.bam/_temp/;
11
12          my $command=$samtools." sort ".$options." -o ".
    $bamFileOut." -T ".$tempPrefix." ".$bamFileIn;
```

The `toolbox::extractOptions` function will create a text version of the hash containing the options for the given tool (first argument). A second optional argument can be provided to specify the separator between the option name and its value. Thus if the second argument is provided as "=", the option output would be "-d=1". Either, in standard it will be "-d 1" (standard is space).

The command line can thus be created.

```
1           #Execute command
2           if(toolbox::run($command)==1)
```

```
3           {
4                 return 1;#Command Ok
5           }
6           else
7           {
8                 toolbox::exportLog("ERROR: samTools::
    samToolsSort : Uncorrectly done\n",0);
9                 return 0;#Command not Ok
10          }
```

Once we have created the command, we can send it to `toolbox::run`, and report the output state (O, 1 or 2).

3.4.1 Adding a perlDoc

A **perlDoc** information has to be added at the end of the module, after the*1;*:

```
1           =head1 NAME
2
3         Package I<samtools>
4
5       =head1 SYNOPSIS
6
7       This package contains the whole modules for the SAMtools
     software
8
9       =head1 DESCRIPTION
10
11      Package SAMtools (Li et al, 2009, http://http://www.
    htslib.org/ ) is a software package for working on SAM and
     BAM files (sorting, selection, merging...)
12
13      =head2 Functions
14
15      =over 4
16
17      =item samToolsSort (Sorts in different ways a BAM file:
    coordinates, random, reads)
18
19      =back
20
21      =head1 AUTHORS
22
23      Intellectual property belongs to IRD, CIRAD, YOUR
    INSTITURE and South Green developpement plateform
24      Written by Cecile Monat, Ayite Kougbeadjo, Marilyne
    Summo, Cedric Farcy, Mawusse Agbessi, Christine Tranchant,
     YOUR NAME and Francois Sabot
25
26      =head1 SEE ALSO
27
28      L<http://toggle.southgreen.fr/>         # SOUTH GREEN
    TOGGLE WEBSITE
29
```

```
30          =cut
```

3.4.2 TIPS

Generally, the fastest and easiest way to create new functions is to copy an existing one (closely related) and to modify it.

Chapter 4

The tests

Tests are important to avoid code regressions and ensure thus a high-quality code.

Five steps are always tested in TOGGLE tests:

1. Can the module be used ?
2. Can the function be called ?
3. Does the function run ?
4. Does the function return the good file list ?
5. Does the function return the expected resulting file ?

You have to create the test file for your module in the **test/modules** directory, named such as **module_test.t**. Thus in the current example, the test file will be **test/modules/samTools_test.t**.

4.1 Basic structure of the test file

```
1  #!/usr/bin/perl
2
3  ################################################################################
4  #
5  # Copyright 2014-2017 IRD-CIRAD-INRA-ADNid-YOUR INSTITUTE
6  #
7  # This program is free software; you can redistribute it and/or modify
8  # it under the terms of the GNU General Public License as published by
9  # the Free Software Foundation; either version 3 of the License, or
10 # (at your option) any later version.
11 #
```

```
12  # This program is distributed in the hope that it will be
      useful,
13  # but WITHOUT ANY WARRANTY; without even the implied
      warranty of
14  # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See
      the
15  # GNU General Public License for more details.
16  #
17  # You should have received a copy of the GNU General Public
      License
18  # along with this program; if not, see <http://www.gnu.org/
      licenses/> or
19  # write to the Free Software Foundation, Inc.,
20  # 51 Franklin Street, Fifth Floor, Boston,
21  # MA 02110-1301, USA.
22  #
23  # You should have received a copy of the CeCILL-C license
      with this program.
24  #If not see <http://www.cecill.info/licences/Licence_CeCILL-
      C_V1-en.txt>
25  #
26  # Intellectual property belongs to IRD, CIRAD and South
      Green developpement plateform for all versions, to ADNid
      for v2 and latter versions, to INRA for v3 and latter
      versions and YOUR INSTITUTE for the current and latter
      versions
27  # Version 1 written by Cecile Monat, Ayite Kougbeadjo,
      Christine Tranchant, Cedric Farcy, Mawusse Agbessi,
      Maryline Summo, and Francois Sabot
28  # Version 2 written by Cecile Monat, Christine Tranchant,
      Cedric Farcy, Enrique Ortega-Abboud, Julie Orjuela-Bouniol
      , Sebastien Ravel, Souhila Amanzougarene, and Francois
      Sabot
29  # Version 3 written by Cecile Monat, Christine Tranchant,
      Laura Helou, Abdoulaye Diallo, Julie Orjuela-Bouniol,
      Sebastien Ravel, Gautier Sarah, and Francois Sabot
30  # Current version written by YOUR NAME and v3 authors
31  #
32  #
      ##############################################################################

33
34  #Will test if samTools module work correctly works correctly
35  use strict;
36  use warnings;
37
38  use Test::More 'no_plan'; #Number of tests, to modify if new
      tests implemented. Can be changed to the true number of
      test instead of 'no_plan'.
39  use Test::Deep;
40  use lib qw(../../modules/);
41
42  ########################################
43  #use of samtools modules ok
```

```
44  ##########################################
45  use_ok('localConfig') or exit; #Test if you can use the
      module localConfig.pm
46  use_ok('samTools') or exit; #Test if you can use the module
      samTools.pm
47
48  can_ok( 'samTools','samToolsSort'); #Test if you can use the
      function samToolsSort from the samTools.pm module
49
50  use localConfig;
51  use samTools;
52
53  ##########################################
54  #Remove files and directory created by previous test
55  ##########################################
56  my $testingDir="$toggle/dataTest/samToolsTestDir";
57  my $cleaningCmd="rm -Rf $testingDir";
58  system ($cleaningCmd) and die ("ERROR: $0 : Cannot remove
      the previous test directory with the command $cleaningCmd
      \n$!\n");
59
60
61  ##########################################
62  #Creation of test directory
63  ##########################################
64  my $makeDirCmd = "mkdir $testingDir";
65  system ($makeDirCmd) and die ("ERROR: $0 : Cannot create the
      new directory with the command $makeDirCmd\n$!\n");
66  chdir $testingDir or die ("ERROR: $0 : Cannot go into the
      new directory with the command \"chdir $testingDir\"\n$!\n
      ");
67
68  ##########################################
69  #Creating the IndividuSoft.txt file
70  ##########################################
71  my $creatingCmd="echo \"samTools\nTEST\" > individuSoft.txt"
      ;
72  system($creatingCmd) and die ("ERROR: $0 : Cannot create the
      individuSoft.txt file with the command $creatingCmd\n$!\n
      ");
73
74  ##########################################
75  #Cleaning the logs for the test
76  ##########################################
77  $cleaningCmd="rm -Rf samTools_TEST_log.*";
78  system($cleaningCmd) and die ("ERROR: $0 : Cannot remove the
      previous log files with the command $cleaningCmd \n$!\n")
      ;
79
80  ##########################################
81  #Identification and linkage to external data, such as
      reference file, if any
82  # Data for tests and other references are in $toggle/data
      folder.
```

```
83  ########################################
84
85  my $bamFile = "$toggle/data/testData/samBam/oneBamUnsorted/
      unsorted.bam";
86
87  #
      ##############################################################################
88  ##Samtools sort
89  #
      ##############################################################################
90
91  #Output file
92  my $bamFileOut = "sorted.bam";
93
94  #execution test
95  is(samTools::samToolsSort($bamFile, $bamFileOut),1,'samTools
      ::samToolsSort');
96
97  # expected output test
98  my $observedOutput = `ls `;
99  my @observedOutput = split /\n/,$observedOutput;
100 my @expectedOutput = ('individuSoft.txt','samTools_TEST_log.
      e','samTools_TEST_log.o','sorted.bam','unsorted.bam');
101
102 is_deeply(\@observedOutput,\@expectedOutput,'samTools::
      samToolsSort - output list');
103
104 # expected output structure
105 my $expectedMD5sum = "c5db29f185507f5433f0c08163a2dc57";
106 my $observedMD5sum=`md5sum sorted.bam`;# structure of the
      test file
107 my @withoutName = split (" ", $observedMD5sum);     # to
      separate the structure and the name of the test file
108 my $observedMD5sum = $withoutName[0];       # just to have
      the md5sum result
109 is($observedMD5sum,$expectedMD5sum,'samTools::samToolsSort -
      output structure');
```

4.2 Precise description of test behavior in TOGGLE

So, if we check the 5 steps

1. Can the module be used ?
   `use_ok('samTools') or exit;`
2. Can the function be called ?
   `can_ok( 'samTools','samToolsSort');`
3. Does the function run ?
   `is(samTools::samToolsIndex($bamFile),1,'samTools::samToolsIndex');`

4. Does the function return the good file list ?
   `is_deeply(@observedOutput,@expectedOutput,'samTools::samToolsSort - output list');`

5. Does the function return the expected resulting file ?
   `is($observedMD5sum,$expectedMD5sum,'samTools::samToolsSort - output structure');`

4.3 Launching the test individually and add it to allTestModules.sh

You can launch the test using the command `prove -v samTools_test.t`. It will invoke the *Perl Test* framework and signal you if your test is working or not.

If working, add it for the whole test system in adding the following command at the end of the **$toggle/test/allTestModules.sh** file:

```
1  ################################## TEST MODULE
2  cmd='prove -v samTools_test.t';
3  echo "
4  ############################################################
5  ############   $cmd   #################
6  ############################################################";
7
8  $cmd;
9
10 echo "
11 ############################################################
12 ############   $cmd DONE   ###############
13 ############################################################";
```

Then launch the complete tests using `sh allTestModules.sh`.

Chapter 5

Creating a new block of code

Once the function and its tests have been created, you can either stand like that, or adding it to the library of bricks we can use in the on the fly pipeline generation... Which is much greater :D

5.1 Already declared variables and other standard stuff

The onTheFly version contains a wide range of already global declared variables and internal variables to in a standard way to know in which step we are. The already declared variables are in the *startBlock.txt* file:

```
1
2  # GLOBAL variables declaration
3
4  ##FASTA associated variables
5  my $fastaFileIn="NA";
6  my $faidxFileOut="NA";
7  my $localFastaFileIn = "NA";
8
9  ##FASTQ associated variables
10 my ($fastqForwardIn, $fastqForwardOut, $fastqReverseIn,
       $fastqReverseOut)=("NA","NA","NA","NA");
11
12 ##SAM associated variables
13 my ($samFileIn, $samFileOut)=("NA","NA"); #Those variables are
       to be used for sam standard but also if the block can
       treat SAM as well as BAM (eg samtools view)
14
15 ## SAI associated variables
16 my ($saiForwardOut, $saiReverseOut)=("NA","NA"); # Use for bwa
       alnBlock, bwaSampe and bwaSamse
17
18 ##BAM associated variables
19 my ($bamFileIn, $bamFileOut)=("NA","NA");
```

```
20 my $listOfBam=();
21
22 ##VCF associated variables
23 my ($vcfFileIn,$vcfFileOut)=("NA","NA");
24
25 ##Intervals/Report associated variables
26 my ($intervalsFile,$tableReport,$vcfSnpKnownFile,$depthFileOut
       );
27
28 ##MpileUp associated variables
29 my ($mpileupOut);
30
31
32 ## INTERNAL VARIABLES
33
34 ###Directory and file variables
35 my ($newDir, $fileWithoutExtension, $extension, $shortDirName,
        @dirList);
36 ### Step variables
37 my ($stepF1, $stepOrder, $stepName, $softParameters);
38 ### Various command variables
39 my ($cleanerCommand, $compressorCommand,$replacementCommand);
```

In a same way, the current directory is already known, such as the previous one (see *previousBlock.txt* and *afterBlock.txt*).

Normally you don't need to change these framework files.

5.2 Create a block

A block is an implementation of a call to the new function you designed. This code will be used latter by *toggleGenerator.pl* to generate the pipeline scripts. You have to create the block file for your fonction in the **onTheFly** directory, named such as **fonctionBlock.txt**. Thus in the current example, the test file will be **onTheFly/samToolsSortBlock.txt**.

Starting with the previous example, let's see what would be the code block associated with

```
1
2 ##########################################
3 # Block for samtools sort
4 ##########################################
5
6 #Correct variable populating
7
8 foreach my $file (@{$fileList}) #Checking the type of files
      that must be BAM
9 {
10     if ($file =~ m/bam$/) # the file type is normally bam
11     {
12         if ($bamFileIn ne "NA") # Already a bam recognized,
      but more than one in the previous folder
```

```
13          {
14              toolbox::exportLog("ERROR : $0 : there are more
    than one single BAM file at $stepName step.\n",0);
15          }
16          else
17          {
18              $bamFileIn = $file;
19          }
20      }
21  }
22
23  if ($bamFileIn eq "NA") #No BAM file found in the previous
    folder
24  {
25      toolbox::exportLog("ERROR : $0 : No BAM file found in
    $previousDir at step $stepName.\n",0);
26  }
27
28  $softParameters = toolbox::extractHashSoft($optionRef,
    $stepName);   # recovery of specific parameters of
    samtools sort
29
30  $bamFileOut = "$newDir"."/"."$readGroup".".SAMTOOLSSORT.bam";
31  samTools::samToolsSort($bamFileIn,$bamFileOut,$softParameters)
    ;   # Sending to samtools sort function
```

As you can see many controls and comments are added.

The first thing done is to check the number of input files. As samToolsSort function will sort only one file at a time, it checks also that the previous folder is not an empty one. Then it recovers the subhash containing the parameters for samtools sort and the arguments are sent to the function.

For creating a new function, the easiest way is to copy a related block, to modify it at convenance then to save it under another name.

5.3 Testing a block

If you block has been created, you need test it. In the **test/blocks** repertory add the test of your fonction at the end of the file corresponding to the tool. Starting with the previous exemple, let's see the test of the block associated with it :

Four steps are done in order to test a block :

1. Removing files and directory created by previous test
2. Creating config file for this test : create an configuration file automatically using the fileConfigurator::createFileConf fonction.
3. Running TOGGLE for the new fonction

4. Checking final results by comparing expected and observed files (content and value output)

```
#####################
## TOGGLE samtools sortsam
#####################

#Input data
my $dataOneBam = "$toggle/data/testData/samBam/oneBamUnsorted/";
my $dataRefIrigin = "$toggle/data/Bank/referenceIrigin.fasta";

print "\n\n#########################################################\n";
print "#### TEST SAMtools sort / no SGE mode\n";
print "#########################################################\n";

# Remove files and directory created by previous test
my $testingDir="$toggle/dataTest/samToolsSort-noSGE-Blocks";
my $cleaningCmd="rm -Rf $testingDir";
system ($cleaningCmd) and die ("ERROR: $0 : Cannot remove the previous test directory with the command $cleaningCmd \n$!\n");

#Creating config file for this test
my @listSoft = ("samToolsSort");
fileConfigurator::createFileConf(\@listSoft,"blockTestConfig.txt");

my $runCmd = "toggleGenerator.pl -c blockTestConfig.txt -d ".$dataOneBam." -r ".$dataRefIrigin." -o ".$testingDir;
print "\n### Toggle running : $runCmd\n";
system("$runCmd") and die "#### ERROR : Can't run TOGGLE for samtools sort";

# check final results
print "\n### TEST Ouput list & content : $runCmd\n";
my $observedOutput = `ls $testingDir/finalResults`;
my @observedOutput = split /\n/,$observedOutput;
my @expectedOutput = ('unsorted.SAMTOOLSSORT.bam');

# expected output test
is_deeply(\@observedOutput,\@expectedOutput,'toggleGenerator - One Bam (no SGE) sorting list ');

# expected output content
$observedOutput=`samtools view $testingDir/finalResults/unsorted.SAMTOOLSSORT.bam | head -n 2 | cut -f4`; # We pick up only the position field
chomp $observedOutput;
my @position = split /\n/, $observedOutput;
$observedOutput= 0;
$observedOutput = 1 if ($position[0] < $position[1]); # the first read is placed before the second one
```

```
41 is($observedOutput,"1", 'toggleGenerator - One Bam (no SGE)
       sorting content ');
```

5.4 Launching the test of block individually and add it to allTestBlock.pl

You can launch the test of block using the command `perl samToolsBlock.pl`. It will invoke the *Perl Test* framework and signal you if your test is working or not.

If working, add it for the whole test system in adding the following command at the end of the **$toggle/test/allTestBlock.pl** file:

```
1 system("perl $toggle/test/blocks/samtoolsBlock.pl") and warn "
      ERROR: $0: Cannot run test for samtoolsBlock.pl  \n$!\n";
```

Then launch the complete tests using `perl allTestBlock.pl`.

5.5 Indicating the input and output

The *softwareFormat.txt* file (root folder) allows the system to verify that the output of the step n are compatible with the input of step n+1.

It is basically informed in the following way:

```
$samToolsSort
IN=SAM,BAM
OUT=SAM,BAM
```

Multiple formats are separated by commas.

5.6 Providing the correct nomenclature

Last step, but not least, the adjustement of the nomenclature... In the code itself, we must respect the format previously described in this manual to call a given function.

However, the users are not du to respect this limitation in the *software.config* file. Thus, they can provide the function for our `samToolsSort` function using the correct nomenclature but also in different ways such as *samtools SORT* e.g.

The transformation/correction is ensured by the `namingConvention::correctName` function:

```
1  sub correctName
2 {
3     my ($name)=@_;
4     my $correctedName="NA";
5     my $order;
6     ## DEBUG toolbox::exportLog("++++++++++++++++$name\n",1);
7     my @list = split /\s/,$name;
```

```
8       $order = pop @list if ($list[-1] =~ m/^\d+/); # This is
        for a repetition of the same step
9       switch (1)
10      {
11      #FOR cleaner
12      case ($name =~ m/cleaner/i){$correctedName="cleaner";} #
        Correction for cleaner step
13
14      #FOR SGE
15      case ($name =~ m/sge/i){$correctedName="sge";} #Correction
         for sge configuration
16
17          #FOR bwa.pm
18          case ($name =~ m/bwa[\s|\.|\-| \/|\\|\|]*aln/i){
        $correctedName="bwaAln"; } #Correction for bwaAln
19          case ($name =~ m/bwa[\s|\.|\-| \/|\\|\|]*sampe/i){
        $correctedName="bwaSampe"} # Correction for bwaSampe
20          case ($name =~ m/bwa[\s|\.|\-| \/|\\|\|]*samse/i){
        $correctedName="bwaSamse"} # Correction for bwaSamse
21          case ($name =~ m/bwa[\s|\.|\-| \/|\\|\|]*index/i){
        $correctedName="bwaIndex"} # Correction for bwaIndex
22          case ($name =~ m/bwa[\s|\.|\-| \/|\\|\|]*mem/i){
        $correctedName="bwaMem"} # Correction for bwaMem
23          ....
24        }
25  }
```

This function will recognize the names based on regular expression, and provide the correct name to the system. It remove spaces, points, dash, slash,... and recognize lower and upper case. To create your own entry, please use the following system

```
1   case ($name =~ m/my[\s|\.|\-| \/|\\|\|]*name/i){
      $correctedName="myName"; } #Correction for myName function
```

Thus for samToolsSort the correction is:

```
1   case ($name =~ m/samtools[\s|\.|\-| \/|\\|\|]*sort/i){
      $correctedName="samToolsSort"; } #Correction for
      samToolsSort function
```

As before, you can copy and modify a closely related line code.

5.7 Last but not least

Please commit all changes individually in YOUR branch and do not forget to push!

```
#Check your current status
git status

#Check the branch you are working on
```

```
git branch

#Perform the commit
git commit -m "My Explicite comment" changedFile

#Push your local branch to GitHub
git push https://github.com/SouthGreenPlatform/TOGGLE-DEV.git branchName
```

Appendix A

Licence