



Dev manual for Toggle

Version 2.0

Authoring by Toggle dev Team

April 19, 2017

# Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
1.1	To whom this manual is addressed . . . . .	2
1.2	General things about the TOGGLE github . . . . .	2
1.2.1	Preparing your working environment - Users registered on the TOGGLE-DEV github . . . . .	2
1.2.2	Preparing your working environment - Users NOT registered on the TOGGLE-DEV github . . . . .	3
1.3	General things about the conventions and nomenclatures in TOGGLE . . . . .	3
<b>2</b>	<b>Creating a new module</b>	<b>4</b>
2.1	Names . . . . .	4
2.2	Requirements and Declaration . . . . .	4
<b>3</b>	<b>Creating a new function</b>	<b>6</b>
3.1	Nomenclature, Indentation and commentaries . . . . .	7
3.2	Basic structure of the function . . . . .	7
3.3	The <code>toolbox::exportLog</code> and <code>toolbox::run</code> functions . . . . .	8
3.3.1	<code>toolbox::exportLog</code> . . . . .	8
3.3.2	<code>toolbox::run</code> . . . . .	8
3.4	The code itself . . . . .	8
3.4.1	TIPS . . . . .	10
3.5	The test . . . . .	10
<b>4</b>	<b>Creating a new block of code</b>	<b>11</b>
4.1	Already declared variables and other standard stuff . . . . .	11
4.2	Text block . . . . .	12
4.3	Indicating the input and output . . . . .	13
4.4	Providing the correct nomenclature . . . . .	13
4.5	Last but not least . . . . .	14
<b>A</b>	<b>Licence</b>	<b>15</b>

# 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 registered 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 first 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 branches will be performed by power users under request on the github. The integration depends on the correct application of the following recommendations, especially tests.

### 1.2.2 Preparing your working environment - Users NOT registered 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/
8 # or modify
9 # it under the terms of the GNU General Public License as
10 # published by
11 # the Free Software Foundation; either version 3 of the
12 # License, or
13 # (at your option) any later version.
14 #
15 # This program is distributed in the hope that it will be
16 # useful,
17 # but WITHOUT ANY WARRANTY; without even the implied warranty
18 # of
19 # 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 plateforme 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 Kougbéadjó,
    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 through 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 an intrinsic 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 maintainers!

### 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 intinsic function that cannot be modified except by maintainers.

## 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 incorrect\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
INSTITUTE and South Green developpement plateforme
24     Written by Cecile Monat, Ayite Kougbéadjó, 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.

## 3.5 The test

I have to fill this part....

## Chapter 4

# Creating a new block of code

Once the function is 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

### 4.1 Already declared variables and other standard stuff

The on the fly version contains a wide range of already declared variables and standard way to know in which step we are. The already declared variables are (in the *startBlock.txt* file):

---

```
1  #Switch and populating variable
2
3  #Standard variables
4
5  my ($softParameters, $newDir, $stepOrder, $stepName,
      $fileWithoutExtention, $fastqForward, $fastqReverse, $samFile,
      $extension, $samFileOut, $shortDirName, $stepF1, $vcfCalled,
      $vcfOut, $listOfBam, $bamFile, $bamFileOut, $cleanerCommand,
      $intervalsFile, $replacementCommand);
6
7  #Variable variables
8  my ($fastq1, $fastq2, $sam, $bam, $vcf);
9  my $fileList=toolbox::readDir($previousDir);
10 my $globalAnalysis=0; #to check if all data are to be treated
    one by one (0) or in group (1)
```

---

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

## 4.2 Text block

A text 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. This is a text file, that must be saved in the *onTheFly* folder to be used.

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

---

```
1 #####
2 # Block for samtools sort
3 #####
4
5 $samFile=0;
6 foreach my $file (@{$fileList}) #Checking the type of files
   that must be SAM or BAM
7 {
8     if ($file =~ m/sam$|bam$/) # the file type is normally sam
       of bam
9     {
10         if ($samFile != 0) # Already a sam of a bam recognized
           , but more than one in the previous folder
11         {
12             toolbox::exportLog("ERROR : $0 : there are more
           than one single SAM/BAM file at $stepName step.\n",0);
13         }
14         $samFile = $file;
15     }
16 }
17
18 if ($samFile eq '0') #No SAM/BAM file found in the previous
   folder
19 {
20     toolbox::exportLog("ERROR : $0 : No SAM/BAM file found in
           $previousDir at step $stepName.\n",0);
21 }
22
23 $softParameters = toolbox::extractHashSoft($optionRef ,
           $stepName); # recovery of specific parameters of
           samtools sort
24
25 samTools::samToolsSort($samFile,$softParameters); # Sending
           to samtools sort function
```

---

As you can see many controls and comments are added.

The first thing done is the checking of number of input file, as *samToolsSort* function will allow only one file to be sort at a time. It check also that the previous folder is not an empty one. Then it recovers the subhash containing the parameters for *samtools sort*, if any, and then send the arguments to the function.

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

### 4.3 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.

### 4.4 Providing the correct nomenclature

Last step, but not least, the adjustment 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 **namConvention::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.

## 4.5 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
```

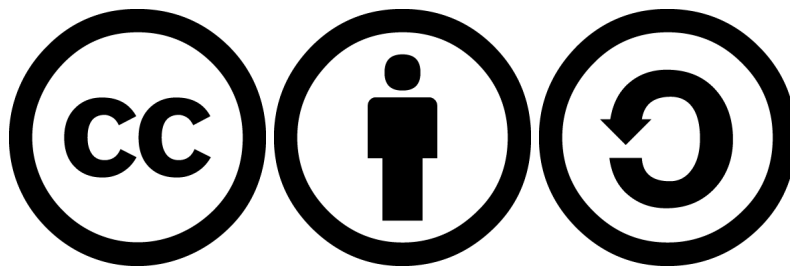
```
#Perform the commit
git commit -m "My Explicite comment" changedFile
```

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



## Appendix A

### Licence



This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license, visit <http://creativecommons.org/licenses/by/4.0/>.