ngasp

Computational solution for performing
next generation analysis of sequence polymorphisms
using NGS data

Research Program : Plant and Animal Genomics
Research Group : Statistical and Population Genomics

Software Engineer: J. Jené

Principal Investigator: S. Ramos
Technical Supervisor: G. Vera

June 9, 2017

Contents

List of Figures

What is the purpose of this book?

I am grateful for the development and introduction of the ngasp system, the computational solution for performing next generation analysis of sequence polymorphisms using NGS data.

It is my sincere hope that either reading this book in its entirety or going through select chapters, will make you understand how to perform your experiments, how to create new calculations and how to continue developing the system.

0.1 What are the objectives of this book?

This book is primarily intended for two audiences. First, it provides information needed to understand the system and to start using it. And second, it provides technical information about how to develop it.

Although this book contains useful information for the End User it is not intended to be used as a End User Manual. This book is aimed at developers that start with the project development.

0.2 How is the information presented?

The book is divided into parts. First of all, it is shown a project overview and a description about how to use the system. Then, it is explained the rules of developing the system and how to develop an example calculation. And at the end, the public web site is presented.

I have done my best to avoid highly technical language. Genetic and scientific terms are defined in the Glossary. For those eager to learn more about a specific topic, I suggest additional reading material in the Bibliography at the end of the book.

0.3 How is the book intended to be used?

You may decide to focus your reading on questions of special interest to you. If you have a broader interest in ngasp development, you may want to read the book from beginning to end.

If the book does not address all of your questions, I would like to hear from you. New questions of broader interest can be added to the next edition. Feel free to contact me with additional questions at joan.jene@cragenomica.es

Part I

Project Overview

Chapter 1

Project Description

After reading this chapter you should be able to download the source code of the project and to identify its main parts.

1.1 Project Repository

The project is stored in the following GitLab's repository: `https://gitlab.cragenomica.es/gvera/ngasp`. To download the source code, open a terminal and execute:

```
$ git config --global http.sslverify false
$ git clone -b develop https://gitlab.bio.crag.local/gvera/ngasp.git , or
$ git clone -b develop https://gitlab.cragenomica.es/gvera/ngasp.git
```

The repository stores all files related with this project: the C++ source code, the public web site, the JavaScript front-end, the source code of external tools, media files, example data and documentation.

The ngasp system is available under the [GNU Lesser General Public License, version 2.1] license.

Let's take a look at the main folder structure.

1.2 Project Folder

Once you downloaded the source code from its repository, you will see these folders:

- **`docker/`**
  This folder contains the Docker files for creating the compiler environment. This environment is used for compiling all project source codes. It is platform independent. The environment can be created using Windows, Mac OS X and Linux (in fact, all operating systems where Docker can be installed in).
- **`media/`**
  This folder contains ngasp resources such as logos, icons and images.
- **`develop/source/backend/`**
  This is the source code of the back-end (the system core).
- **`develop/webapp/`**
  This is the folder of the front-end (web application).
- **`develop/webapp/tmp/`**
  This is the temporary folder used by the back-end for storing experiment results and messages.
- **`develop/source/samtoolslib/`**
  This is the source code of samtools with some modifications (search for this tag: "//!").

- **`develop/source/seqan/`**
  This is the source code of the seqan library.
- **`develop/repository/`**
  It contains some example experiments and pipelines.
- **`develop/examples/`**
  This folder contains data example files. Only for development purposes.
- **`/develop/tests/`**
  This folder contains development scripts for testing purposes.
- **`doc/`**
  This folder contains development documentation about the project.
- **`gradle/`**
  This folder is used by Jenkins.
- **`gradlew`**
  This files is used by Jenkins on Linux systems.
- **`gradlew.bat`**
  This files is used by Jenkins on Windows systems.
- **`LICENSE.txt`**
  This is the license text for this project.
- **`ngasp.xcodeproj`**
  It contains the XCode configuration for opening the ./source/backend.
- **`public-web/`**
  This folder contains a copy of the website:
  `https://bioinformatics.cragenomica.es/projects/ngaSP`
- **`readme.txt`**
  This file.
- **`release/`**
  This folder contains the final application folder.
  This folder is compressed and stored on the public web for being downloaded by Jenkins.
  Attention: Jenkins copies extra files to this folder. So, it is not usable unless you get this folder from the download page (after Jenkins tasks are done).
  These extra files are: the folder `webapp` that must be inside the `cm` and `lm` folders and the `repository` folder that must be in the root folder with `cm`, `lm`, `step1` and `step2` folders.
- **`tools/`**
  It contains some development tools.
- **`./.git`**
  This folder owns to Git.
- **`./.gitignore`**
  It excludes files and folders from Git transactions.

Chapter 2

Development Environment

After reading this chapter you should be able to identify all docker containers (Figure 2.1) and to prepare your workstation to be able to develop the project.

You will see that the use of Docker simplifies the installation process due to all tools and required libraries will be installed automatically.

2.1 The Compiler Environment Container

The compiler environment is the Docker container (Figure 2.1) used by Jenkins for compiling the source code of this project. But it is also the base of the development environment container that is used for developing the project in a workstation.

So, let's start taking a look to the compiler environment container. To do so, it is necessary to install Docker in your workstation. Follow installation instructions from `https://docs.docker.com/engine/installation`. This is a Docker example installation on Centos 7:

```
$ sudo yum update
$ tee /etc/yum.repos.d/docker.repo <<-'EOF'
[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/main/centos/7/
enabled=1
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
EOF
$ sudo yum install docker-engine   or    sudo yum install docker
$ sudo systemctl enable docker.service
$ sudo systemctl start docker
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
$ sudo newgrp docker
```

It is also needed to install Docker Compose, from `https://docs.docker.com/compose/install`:

```
$ curl -L https://github.com/docker/compose/releases/download/1.9.0-rc1/docker-compose-`uname -s
    `-`uname -m` > /usr/bin/docker-compose
$ chmod +x /usr/bin/docker-compose
```

And finally, create the the compilation environment with the following tools and libraries:

- **c++ compiler**: Required by the back-end and its c++ external tools.
- **make**: Required by the back-end and its c++ external tools.
- **bzip2**: Required by unpacking downloaded files.

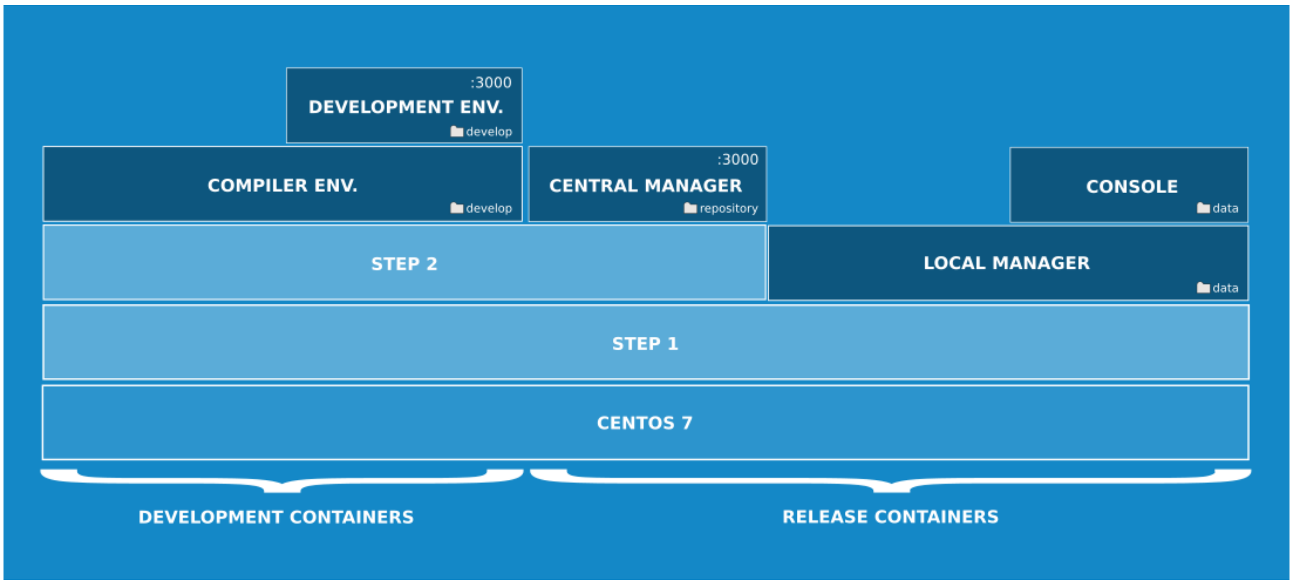


Figure 2.1: Docker Containers

- **nodejs**: Required by the front-end.
- **gsl**: Required by the back-end.
- **zlib**: Required by the back-end and its c++ external tools.
- **htslib**: Required by the back-end.
- **boost**: Required by the back-end.
- **openmpi**: Required by the *GHcaller*.

Run:

```
$ cd <ngasp_folder>/docker
$ docker-compose -f compose.yml build
$ docker-compose -f compose.yml up
$ docker-compose -f compose.yml start
```

To ensure that all docker images are created, run the following command:

```
$ docker images
REPOSITORY             TAG                 IMAGE ID            CREATED             SIZE
compiler-environment   latest              22d3ff071e43        8 weeks ago         2.409 GB
step2                  latest              1b8dbedbb3a3        8 weeks ago         1.58 GB
step1                  latest              d6374b8c2979        8 weeks ago         1.474 GB
centos                 latest              980e0e4c79ec        3 months ago        196.7 MB
$
```

To ensure that all dockers are created, run the following command:

```
$ docker ps -a
CONTAINER ID IMAGE                COMMAND     CREATED        STATUS                        NAMES
f627f71f2a3e compiler-environment "/bin/bash" 3 minutes ago Exited (0) About a minute ago
    compiler-environment
4eb7d06c6110 step2                "/bin/bash" 3 minutes ago Exited (0) About a minute ago step2
4d2c2abe6407 step1                "/bin/bash" 3 minutes ago Exited (0) About a minute ago step1
283f571a2bfb centos               "/bin/bash" 9 weeks ago   Exited (0) 9 weeks ago
    elegant_snyder
$
```

Jenkins will use this container for compiling the source code, running:

```
cd <ngasp_folder>/docker
docker-compose -f compose.yml build
docker run --rm -v /home/ci_ngasp/ngasp/develop:/develop compiler-environment /develop/
    compile_all.sh
```

2.2 Development Environment Container

The Development Environment container (Figure 2.1) is used for developing the project in a workstation.

- First, build the Compiler Environment: (See Section 2.1)

```
$ cd <ngasp_folder>/docker
$ docker-compose -f compose.yml build
```

- Build the Development Environment container:

```
$ cd <ngasp_folder>/docker/development-environment
$ docker build -t development-environment .
```

- Accessing the Development Environment container, on Linux:

```
$ cd <ngasp_folder>
$ docker run -it \                              # Interactive mode
             --rm \                             # Remove the container when exiting it
             --privileged \                     # Enabled for debugging the application
             -p 3000:3000 \                     # localhost:3000 enabled on the host
             -e DISPLAY=$DISPLAY \              # GUI apps enabled
             -v `pwd`/develop:/develop \        # Access to the source code on R/W mode
             -v /tmp/.X11-unix:/tmp/.X11-unix \ # GUI apps enabled
             development-environment \          # Use this image
             /bin/bash                          # Execute this application

# Example (set your own paths):
$ docker run -it --rm --privileged -p 3000:3000 -e DISPLAY=$DISPLAY -v `pwd`/develop:/
    develop -v /tmp/.X11-unix:/tmp/.X11-unix development-environment /bin/bash
```

- Accessing the Development Environment container, on Mac OS X:

```
$ cd <ngasp_folder>
$ curl -o XQuartz-2.7.11.dmg https://dl.bintray.com/xquartz/downloads/XQuartz-2.7.11.dmg -
    LOk
$ sudo hdiutil attach XQuartz-2.7.11.dmg
$ sudo installer -package /Volumes/XQuartz-2.7.11/XQuartz.pkg -target /
$ sudo hdiutil detach /Volumes/XQuartz-2.7.11/
$ rm XQuartz-2.7.11.dmg
# Now, log out and log in.
$ open -a XQuartz                               # Security:
                                                # [x] Authenticate connections
                                                # [x] Allow connections from net clients
$ ip=$(ifconfig en0 | awk '$1=="inet" {print $2}')
$ xhost + $ip
$ docker run -it \                              # Interactive mode
             --rm \                             # Remove the container when exiting it
             --privileged \                     # Enabled for debugging the application
             -p 3000:3000 \                     # localhost:3000 enabled on the host
             -e DISPLAY=$ip:0 \                 # GUI apps enabled
             -v `pwd`/develop:/develop \        # Access to the source code on R/W mode
             -v /tmp/.X11-unix:/tmp/.X11-unix \ # GUI apps enabled
             development-environment \          # Use this image
             /bin/bash                          # Execute this application
```

- Opening the back-end source code with Netbeans:

```
$ netbeans &
```

Then, click on *Open project...* and open `/develop/source/backend`

Chapter 3

Project Modules

After reading this chapter you should be able to edit and compile the core of the ngasp project and any of its external tools.

3.1 Back-End

The back-end is the core of this project and it is written in c++ language.

For developing this module follow this instructions:

- **Requirements**: Create the Development Environment in your workstation (See Chapter 2).
- **Container**: Use the Development Environment container (See Section 2.2):
- **Locate the source code**:

  ```
  $ cd /develop/source/backend
  ```

- **Compile**:

  You can use Gradle:

  ```
  $ cd /develop
  $ gradle build -b .gradle/build_backend.gradle
  # output here: /develop/webapp/bin/ngasp
  ```

  You can use NetBeans, also:

  – Open the project: *File > Open Project...* : `/develop/source/backend`
  – Press *Shift + F11* for cleaning and building the project.
  – The outpt binary is created here: `/develop/webapp/bin/ngasp`

These are the back-end main file folders:

- **`commands`**: They are keywords that realize basic actions inside the application environment. For example, for creating a new memory variable, for opening an ngasp script, for printing a variable value, ... You will find the source code of all these calculations here:
  `./ngasp/develop/source/backend/\glspl{calculation}s/Calc`
- **`calculations`**: They are c++ functions defined by the Calculation Developer. They can be calls to external applications, also. You will find the source code of all these calculations here:
  `./ngasp/develop/source/backend/command/CMD`

- **`data_manager`**: It is a set of data classes: Such as: CDataInt, CDataDouble, CDataIntVector, ... It is also the manager of these data classes. You will find the source code of all these Data here:
  `./ngasp/develop/source/backend/data_manager/Data`
- **`ipc`**: It manages the way that a *Local Manager* talks with a back-end process.
- **`language`**: All application texts and messages are written here. commands and calculations names, also. Rename commands here:
  `./ngasp/develop/source/language/CStringTable.cpp`
- **`messages`**: It manages the application error, warning, debug, messages,...

3.2 Samtoolslib

The source code of samtools has been included inside the ngasp project. It has been turned into a c++ library and the back-end includes it.

For developing this module follow this instructions:

- **Requirements**: Create the Development Environment in your workstation (See Chapter 2).
- **Container**: Use the Development Environment container (Figure 2.1):

```
$ docker run -ti --rm -v <absolute_path>/ngasp/develop:/develop -p 3000:3000 development-
    environment /bin/bash
```

- **Locate the source code**:

```
$ cd develop/source/samtoolslib/
```

- **Compile**: Use instructions that you will find here: `cat /develop/compile_all.sh`

3.3 Seqan

The source code of seqan has been included inside the ngasp project. But it is used by one or two calculations that are not used yet.

3.4 External tools not included in the project's repository

These external tools are stored in its own repository. The continuous integration environment downloads them and it compiles them.

3.4.1 fastaconvtr

The last release can be downloaded from
`https://gitlab.cragenomica.es/jjene/fastaconvtr/repository/archive.zip?ref=master`
Or, it can be cloned, running:

```
git clone https://gitlab.bio.crag.local/jjene/fastaconvtr.git
```

3.4.2 PFcaller

The last release can be downloaded from
`https://gitlab.cragenomica.es/jjene/PFcaller/repository/archive.zip?ref=master`
Or, it can be cloned, running:

```
git clone https://gitlab.bio.crag.local/jjene/PFcaller.git
```

3.4.3 Fasta2TFasta

The last release can be downloaded from
`https://gitlab.cragenomica.es/jjene/fasta2tfasta/repository/archive.zip?ref=master`
Or, it can be cloned, running:

```
git clone https://gitlab.bio.crag.local/jjene/fasta2tfasta.git
```

3.4.4 pGHcaller

The last release can be downloaded from
`https://gitlab.cragenomica.es/jnavarro/pghcaller/repository/archive.zip?ref=master`
Or, it can be cloned, running:

```
git clone https://gitlab.bio.crag.local/jnavarro/pghcaller.git
```

3.4.5 mstatspop

The last release can be downloaded from
`https://gitlab.cragenomica.es/jjene/mstatspop/repository/archive.zip?ref=master`
Or, it can be cloned, running:

```
git clone https://gitlab.bio.crag.local/jjene/mstatspop.git
```

3.4.6 gzapp (with last version of zutil & zindex)

The last release can be downloaded from
`https://gitlab.cragenomica.es/jjene/gzapp/repository/archive.zip?ref=master`
Or, it can be cloned, running:

```
git clone https://gitlab.bio.crag.local/jjene/gzapp.git
```

3.4.7 tfaviewer

The last release (Figure 3.1) can be downloaded from
`https://gitlab.cragenomica.es/jjene/tfaviewer/repository/archive.zip?ref=master`
Or, it can be cloned, running:

```
git clone https://gitlab.bio.crag.local/jjene/tfaviewer.git
```

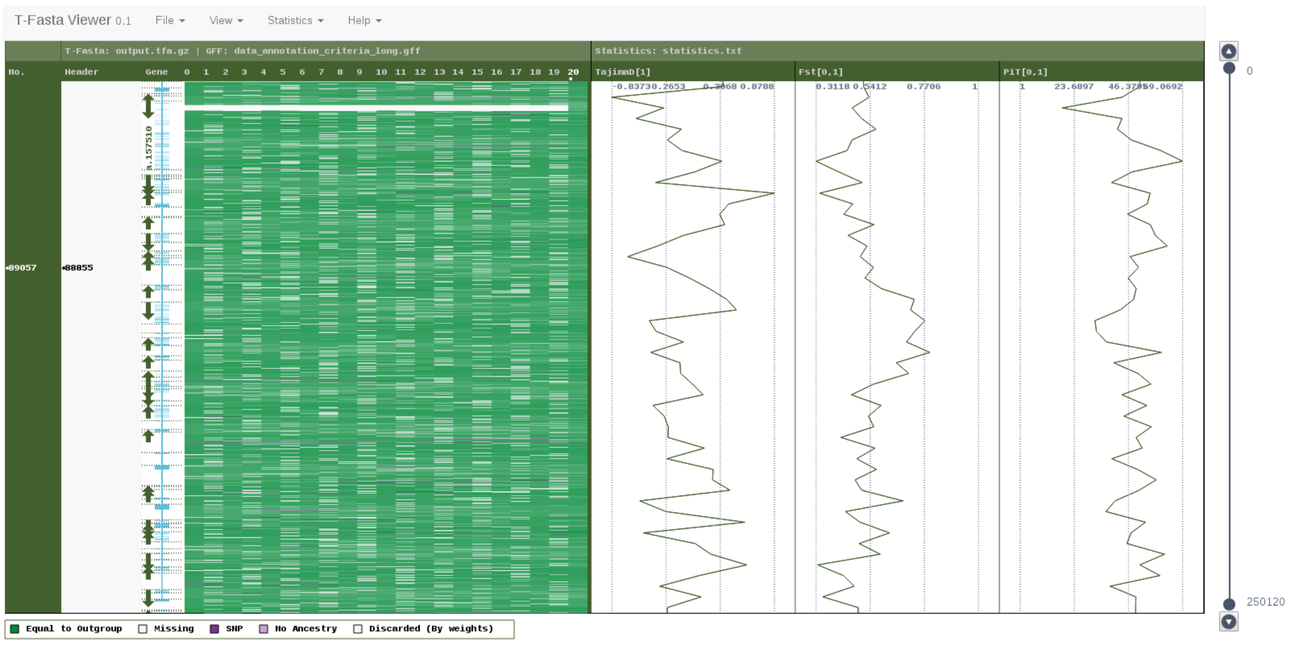


Figure 3.1: T-Fasta Viewer

Chapter 4

Front-End

After reading this chapter you will be able to identify the different modules of the front-end and to start it.

4.1 Front-End Architecture

The ngasp's front-end is developed with HTML5 and JavaScript.

For developing the front-end follow this instructions:

- **Requirements**: Create the Development Environment in your workstation (See Chapter 2).
- **Container**: Use the Development Environment container (Figure 2.1):

  ```
  $ docker run -ti --rm -v <absolute_path>/ngasp/develop:/develop -p 3000:3000 development-
      environment /bin/bash
  ```

- **Locate the source code**:

  ```
  $ cd develop/webapp/
  ```

- **Run**:

  ```
  $ ./start_ngasp.sh
  ```

- **Test**: Open a web navigator in the Host to `localhost:3000`
- **Stop**:

  ```
  $ ./stop_ngasp.sh
  ```

Chapter 5

System Layers

Description

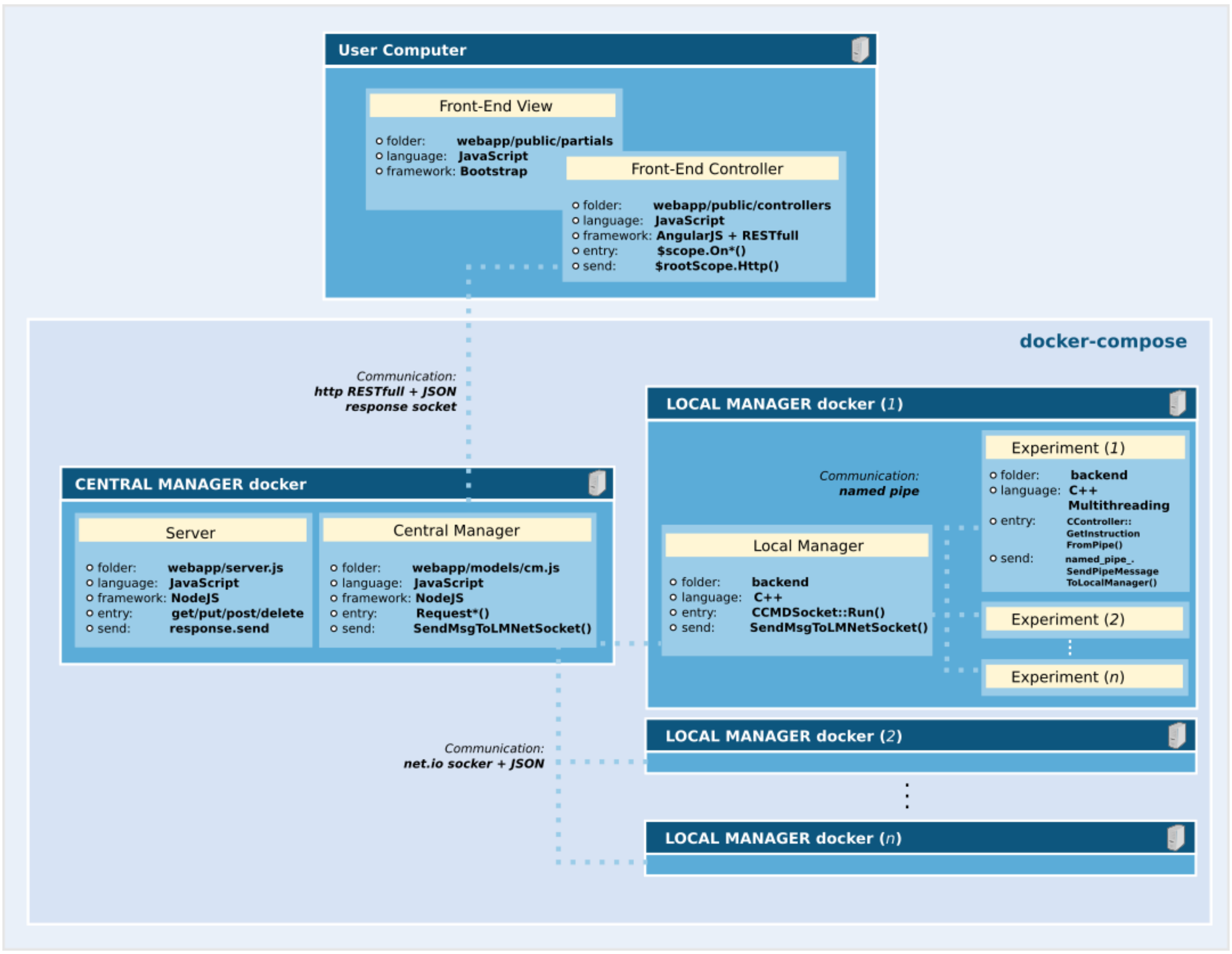


Figure 5.1: Layers

Part II

System Usage

Chapter 6

First Steps

After reading this chapter you, as a End User, will be able to run ngasp in all its modes: Graphical Mode, Command Line Mode and Interactive Mode.

6.1 Download the Application

You can download ngasp from its web site. Follow next steps:

- Navigate to `https://bioinformatics.cragenomica.es/projects/ngaSP`
- Open `Software > Download`.
- Press the ngasp download button.
- Scroll to the end of the list and download the last release.
- Uncompress the package into your workstation. From now on, this folder will be called `<ngasp_folder>`.

6.2 Install the Application

Use the following steps for installing Docker `https://docs.docker.com/engine/installation` `https://docs.docker.com/compose/install`:

1. First of all, install Git if it is not already installed in your computer:

   Example of Installation on Centos 7:

   ```
   $ sudo yum install git
   ```

   Example of Installation on Mac OS X / Windows:

   ```
   Install it from http://git-scm.com/download/mac
   ```

2. Install Docker and its requirements:

   Example of Installation on Centos 7:

   ```
   $ sudo yum update
   $ tee /etc/yum.repos.d/docker.repo << EOF
   [dockerrepo]
   name=Docker Repository
   baseurl=https://yum.dockerproject.org/repo/main/centos/7/
   enabled=1
   gpgcheck=1
   ```

```
gpgkey=https://yum.dockerproject.org/gpg
EOF
$ sudo yum install docker-engine
or
sudo yum install docker
$ sudo systemctl enable docker.service
$ sudo systemctl start docker
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
$ sudo newgrp docker

$ curl -L https://github.com/docker/compose/releases/download/1.13.0-rc1/docker-compose-
`uname -s`-`uname -m` > /usr/bin/docker-compose
$ chmod +x /usr/bin/docker-compose
```

Example of Installation on Mac OS X:

- Navigate to `http://www.docker.com/products/overview`
- Press the *Download* button under the *MAC* platform.
- Open the downloaded *Docker.dmg* file.
- Drag and Drop *Docker* to *Applications*.
- Click on *Docker*.
- Do *Next* and *OK* until Docker informs that it is installed.

Example of Installation on Windows:

- Install Docker from https://docs.docker.com/docker-for-windows/
- Install Docker-Compose from https://docs.docker.com/compose/install/

3. Install XQuartz (only Mac OS X):

```
$ cd <ngasp_folder>
$ curl -o XQuartz-2.7.11.dmg https://dl.bintray.com/xquartz/downloads/XQuartz-2.7.11.dmg
-LOk
$ sudo hdiutil attach XQuartz-2.7.11.dmg
$ sudo installer -package /Volumes/XQuartz-2.7.11/XQuartz.pkg -target /
$ sudo hdiutil detach /Volumes/XQuartz-2.7.11/
$ rm Xquartz-2.7.11.dmg
# Now, log out and log in.
$ open -a XQuartz
# Security:
# [x] Authenticate connections
# [x] Allow connections from net clients
```

4. Download the source code:

Example of Installation on Centos 7 / Mac OS X:

```
$ git clone https://github.com/CRAGENOMICA/ngasp-workshop01.git
$ cd <ngasp_folder>
$ git checkout develop
```

Example of Installation on Windows:

```
X:\>git clone https://github.com/CRAGENOMICA/ngasp-workshop01.git
X:\>cd <ngasp_folder>
X:\ngasp_folder>git checkout develop
```

5. Install ngasp: Example of Installation on Centos 7 / Mac OS X:

```
$ cd <ngasp_folder>
$ install.sh
```

Example of Installation on Windows:

```
X:\>cd <ngasp_folder>
X:\ngasp_folder>install.bat
```

6.3 Launch the Application in Graphical Mode

To open the application in graphical mode, run the following instructions:

- Create the Docker containers (Figure 2.1):

```
$ docker-compose -f compose.yml up
```

- Open: `http://localhost:3000`.

All modifications done to pipelines and experiments will be stored in the `<ngasp_folder>/repository` folder.

6.4 Launch the Application in Command Line Mode

To open the application in command line mode, run:

- Open the *command-line* docker (it has all compiled binaries and all required libraries) (Figure 2.1):

```
$ docker-compose -f compose.yml run --rm command-line
```

- Execute ngasp:

```
$ ./bin/ngasp help
```

- Close the *command-line* docker:

```
$ exit
```

- Open again the previous Docker:

```
$ docker ps -a                       # For getting the just created docker Id
CONTAINER ID        IMAGE            ...
9348fd06563b        command-line     ...
$ docker start 9348fd06563b          # For opening again the same container
$ docker attach 9348fd06563b
```

- Remove the Docker:

```
$ docker ps -a                     # For getting the docker list Ids
CONTAINER ID        IMAGE               COMMAND              ...
9348fd06563b        command-line        "/bin/bash"          ...
$ docker stop 9348fd06563b
$ docker rm 9348fd06563b
```

All docker modified files will be restored after you remove the Docker. So, use the `/app/data/` folder to store changes permanently.

6.5 Launch the Application in Interactive Mode

The application can be also executed in interactive mode. It means that you can open a console window, run the application and it will offer you a prompt where you can execute all the commands that you want. In this mode you can create variables and store their values in memory until you close the interactive mode.

- Open the *command-line* docker (it has all compiled binaries and all required libraries) (Figure 2.1):

```
$ docker-compose -f compose.yml run --rm command-line
```

- Execute the interactive mode:

```
$ bin/ngasp -

Connected: 2016-12-22 08:45:55
Welcome to ngaSP.


ngaSP>
```

- Close the interactive mode:

```
$ bin/ngasp -

Connected: 2016-12-22 08:45:55
Welcome to ngaSP.


ngaSP> exit
Connection to ngaSP closed.
```

- Close the *command-line* docker:

```
$ exit
```

Chapter 7

Commands

After reading this chapter you will know the list of ngasp commands, how to get information about them and some examples of how to use them.

7.1 Command Line Commands

This is the list of ngasp commands with a brief description:

- **`help`**: This command shows the application help.
- **`version`**: This command show the application version number.
- **`exit`**: Stop the application while it is in stdin mode.
- **`load-instructions-file`**: This command open and executes an instructions file.
- **`verbose`**: This function increases / decreases the verbose level of logs.
- **`dry-run`**: This command enables and disables the dry-run option. When activated, all instructions are checked but not executed.
- **`memory-info`**: This command shows information about all Data Manager objects in memory.
- **`open-data-file`**: This command opens a file with data: a fasta file, an ms file, ...
- **`delete`**: This command deletes objects from memory.
- **`history`**: This command manages the list of executed instructions.
- **`dim`**: This command creates variables of different types.
- **`calc`**: This command defines calculations of different types.
- **`set-value`**: This command assigns values to variables.
- **`with`**: Selects the memory object to use.
- **`end-with`**: End of a 'with' block.
- **`print`**: Print memory information.
- **`run-calculation`**: This command runs calculations.
- **`add`**: This command adds a file to the Data Manager.
- **`log`**: This command shows the log file content.
- **`if`**: Start Condition.

- **`else`**: Else Condition.
- **`endif`**: End Condition.
- **`for`**: Loop start.
- **`end-for`**: End of loop.
- **`constant`**: Defines a constant.
- **`reserve`**: This command allocates memory for a table.
- **`syntax`**: This command adds to your editor the capability of reading ngasp files.
- **`tcp`**: Sets the application in Local Manager Execution Mode. It can receive commands via TCP protocol.
- **`pipe`**: Sets the application in Worker Execution Mode. It can receive commands via named pipe.
- **`output`**: This command redirects the output to a file or to the screen.
- **`execute`**: This command executes an external application or system command.
- **`save-state`**: This command saves the Data Manager (DM) data into a file.
- **`reset-memory`**: This command clears the system memory.
- **`matappend`**: This command appends a value, a vector or a matrix to a matrix.

For further information about each command, use the command *help* with the option *-n* and the *command name*. For example:

```
$ ./ngasp help -n mstatspop
NAME
      print - Print memory information.

SYNOPSIS
      print [-n value] [-t value] [-m value] [-e]

DESCRIPTION
      -n, --name
            Name of the memory object to be printed.

      -t, --text
            Print user text.

      -m, --mode
            normal: value is shown in a table.
            json: value is shown as JSON format.
            ngasp: value is shown as a variable definition.
            value: only the value is shown.

      -e, --eol
            Writes an end of line at the end.
```

Chapter 8

Calculations

After reading this chapter you will know the list of ngasp calculations, how to get information about them and some examples of how to use them.

8.1 Calculations

This is the list of ngasp calculations with a brief description:

- **`calc-mstatspop`**: Variability AnalysIs of multiple populations: calculations and estimation of statistics and neutrality tests.
- **`calc-fasta-to-tfasta`**: It converts a Fasta file to a TFasta file.
- **`calc-gc-content`**: *GC Content* (or guanine-cytosine content) is the percentage of nitrogenous bases on a DNA molecule that are either guanine or cytosine. *GC Content* is usually expressed as a percentage value, but sometimes as a ratio.
- **`calc-execute`**: This command executes an external application or system command.
- **`calc-bam-to-bai`**: It creates a BAM index from a BAM file.
- **`calc-bam-chromosomes`**: Get BAM chromosomes.
- **`calc-bam-to-mpileup`**: BAM To Mpileup Conversor.
- **`calc-snp-caller`**: SNP Caller.
- **`calc-concat-files`**: It concatenates two text files.
- **`calc-collect-data-columns`**: Collect data columns.
- **`calc-boxplot-values`**: This calculation receives an array and returns boxplot values.
- **`calc-gVCF-to-tFasta`**: gVCF to TFasta Conversor.
- **`calc-npstat`**: It executes the npstat application.
- **`calc-cut-bed`**: It cuts a BED file.
- **`calc-get-chrom`**: It gets chromosome names from BAM index file.
- **`calc-example`**: This is an example calculation.

For further information about each calculation, use the command *help* with the option *-n* and the *calculation name*. For example:

```
$ ./ngasp help -n calc-bam-to-bai
```

Or, you can move the mouse over the calculation box in *Graphical Mode* and you will get a pop-up window with calculation information.

Chapter 9

Data Types

After reading this chapter you will know the list of ngasp data types.

9.1 Basic Types

This is the list of ngasp file data types.

- **`bool`***: Implemented by the `CDataBoolean` class. 0 is False, 1 is True.
- **`char`***: Implemented by the `CDataChar` class. It is the c++ *char* type.
- **`double`***: Implemented by the `CDataDouble` class. It is the c++ *double* type.
- **`float`***: Implemented by the `CDataFloat` class. It is the c++ *float* type.
- **`int`***: Implemented by the `CDataInt` class. It is the c++ *int* type.
- **`int64`***: Implemented by the `CDataInt64` class. It is the c++ *long int* type.
- **`string`**: Implemented by the `CDataStdString` class. It is a C++ std::string type.
- **`string_vector`**: Implemented by the `CDataStdStringVector` class. It is a vector of C++ std::string type.
- **`string_set`**: Implemented by the `CDataStringSet` class. It is a SeqAn string type.

* These data types can be defined also as vector, matrix and cube (a vector of matrices). For example:

- **int**: 10
- **int_vector**: "10,20,30"
- **int_matrix**: "10,20,30;10,20,30"
- **int_cube**: "10,20,30;10,20,30|10,20,30;10,20,30"

9.2 File Types

This is the list of ngasp file data types:

- **`text_file`**: Implemented by the `CDataTextFile` class. It has a file name, only.

- **`bcf_file`**: Implemented by the `CDataBCF` class. Htslib file.
- **`bam_file`**: Implemented by the `CDataBam` class. It has a file name, only.
- **`bam_index_file`**: Implemented by the `CDataBamIndex` class. It has a file name, only.
- **`sam_file`**: Implemented by the `CDataSam` class. It has a file name, only.
- **`fasta_file`**: Implemented by the `CDataFasta` class. It has a file name, only. (Htslib file commented).
- **`gff_file`**: Implemented by the `CDataGFF` class. It has a file name, only.
- **`gtf_file`**: Implemented by the `CDataGTF` class. It has a file name, only.
- **`t_fasta_file`**: Implemented by the `CDataTFasta` class. It has a file name, only.
- **`g_fasta_file`**: Implemented by the `CDataGFasta` class. It has a file name, only.
- **`mpileup_file`**: Implemented by the `CDataMpileup` class. Htslib file.
- **`snp_file`**: Implemented by the `CDataSNP` class. It has a file name, only.
- **`vcf_file`**: Implemented by the `CDataVCF` class. It has a file name, only.

Chapter 10

ngasp Scripts

After reading this chapter you will be able to create ngasp scripts for executing complex tasks.

10.1 Hello World Sample

First, let's create a Hello World in the interactive mode:

```
$ bin/ngasp -

Connected: 2016-12-22 08:45:55
Welcome to ngaSP.


ngaSP> print --text "Hello World!" --eol
Hello World!
ngaSP>
```

Let's increment the complexity. Let's create a Hello World using variables. The following example will create a variable named *message* and it will be assigned the value "Hello World!".

Note:
`mem` is used for showing the created variable in memory.
`set-value` is used for assigning a value to the variable message.
`print` is used for printing the variable value but note that it can show the value in different ways.

```
$ bin/ngasp -
ngaSP> dim -n message --as string
ngaSP> mem
mem
---------------------------------------------------------------------------------
 Name                                      Type                           Size
---------------------------------------------------------------------------------
 message                                   string                      0 bytes
---------------------------------------------------------------------------------
ngaSP> set-value --to message --eq "Hello World!"
ngaSP> print -n message
 message : string
---------------------------------------------------------------------------------
 Attribute                 Type                       Value
---------------------------------------------------------------------------------
 value                     string                     Hello World!
---------------------------------------------------------------------------------
ngaSP> print -n message --mode value --eol
Hello World!
ngaSP>
```

Note: Sometimes you will need to remove variables. You can do:

```
ngaSP> del -n message
ngaSP>
```

Now, we can write the same example into a ngasp script:

```
$ vi example.ngasp
dim -n message --as string
set-value --to message --eq "Hello World!"
print -n message --mode value --eol
exit
$ ./bin/ngasp load -i example.ngasp
Hello World!
```

10.2 Loops and Conditionals

Using the previous example, let's create a more complex script. Let's write "Hello World!" ten times:

```
$ vi example.ngasp
dim -n message --as string
dim -n i       --as int

set-value --to message --eq "Hello World!"
set-value --to i       --eq 0

for --ite i --from 0 --to 10 --inc 1
   print -n message --mode value --eol
end-for
exit
$ ./bin/ngasp load -i example.ngasp
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
Hello World!
```

Using the previous example, let's print also the iteration number and let's omit iterations 7 and 8:

```
$ vi example.ngasp
dim -n message --as string
dim -n i       --as int

set-value --to message --eq "Hello World!"
set-value --to i       --eq 0

for --ite i --from 0 --to 10 --inc 1
  if --ref i --ne 7 --and --ref i --ne 8
    print -n i --mode value
    print --text ": " --mode value
    print -n message --mode value --eol
  endif
end-for
exit
$ ./bin/ngasp load -i example.ngasp
0: Hello World!
1: Hello World!
2: Hello World!
3: Hello World!
4: Hello World!
5: Hello World!
6: Hello World!
9: Hello World!
```

Note: You can use loops inside loops. Even, you can iterate throw vector elements (similar to a foreach command):

```
$ vi example.ngasp
dim -n valors      --as int_vector
dim -n one_value   --as int

set --to valors --eq "0,1,2,3,4"

for --data valors --ite one_value --from 0 --to valors.reg_length --inc 1
  print -n one_value --mode value --eol
end-for

exit
$ ./bin/ngasp load -i example.ngasp
0
1
2
3
4
```

10.3 Calculation Call in Command Line Mode

Let's see how to call the *GC Content* calculation. This calculation, receives a vector with a DNA sequence and a boolean for indicating if we prefer to get the result as a percentage or as a ratio. The calculation returns a result value:

```
$ vi example.ngasp
verbose --level debug

const --name FALSE --by 0
const --name TRUE  --by 1

dim -n sequence       --as char_vector
dim -n get_percentage --as bool
dim -n total          --as float

set-value --to sequence       --eq "A,T,T,C,G,G,C,T,A"
set-value --to get_percentage --eq FALSE

calc -n c1 --as calc-gc-content
set --to c1.inputs --eq "sequence,get_percentage"
set --to c1.outputs --eq "total"
run -n c1

print -n total --mode json --eol
exit
$ ./bin/ngasp load -i example.ngasp
Verbose Level: debug
00:00:00
$ constant --name FALSE --by 0
00:00:00
$ constant --name TRUE --by 1
00:00:00
$ dim -n sequence --as char_vector
00:00:00
$ dim -n get_percentage --as bool
00:00:00
$ dim -n total --as float
00:00:00
$ set-value --to sequence --eq "A,T,T,C,G,G,C,T,A"
00:00:00
$ set-value --to get_percentage --eq 0
00:00:00
$ calc -n c1 --as calc-gc-content
00:00:00
$ set-value --to c1.inputs --eq "sequence,get_percentage"
00:00:00
$ set-value --to c1.outputs --eq "total"
00:00:00
$ run-calculation -n c1
00:00:00
$ print -n total --mode json --eol
{"id":"total","type":"float","value":"1.25"}
```

```
00:00:00
$ exit
00:00:00
```

Note: It is posible to change the verbose level to debug in order to see each executed line and the its lapse time. Note also that the result is printed as JSON using the *print json mode*.

10.4 Calculation Call in Graphical Mode

Let's see how to call the *GC Content* calculation using the graphical interface.

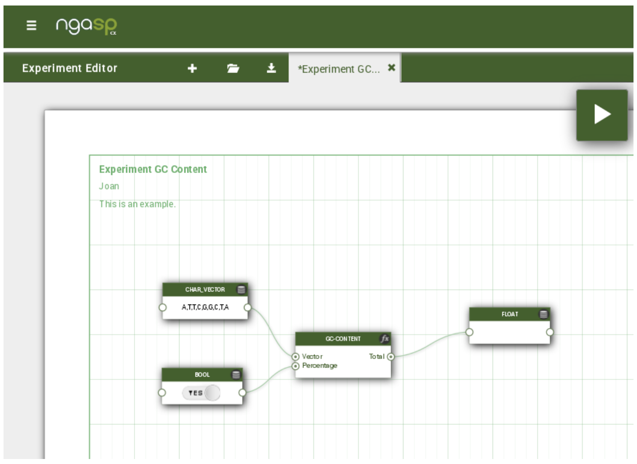
We are going to create the graph of Figure 10.1a.

- Right click over the document. A menu will appear.
- Click on *Add calculation* and *GC Content*. The calculation box will appear.
- Right click over the *Vector* input. A menu will appear.
- Click on *Add variable*. A variable box will appear connected to this input.
- Right click over the *Percentage* input. A menu will appear.
- Click on *Add variable*. A variable box will appear connected to this input.
- Right click over the *Total* output. A menu will appear.
- Click on *Add variable*. A variable box will appear connected to this output.
- Double left click over every input variable box and fulfill with *Vector*: A,A,A,T,C,G,G,A and *Percentage*: NO
- Click the Save button. A pop-up window will appear for saving this experiment.
- Fulfill with *Name*: Experiment *GC Content*; *Author*: Your name; *Description*: This is an example.
- Press the *Save* button.

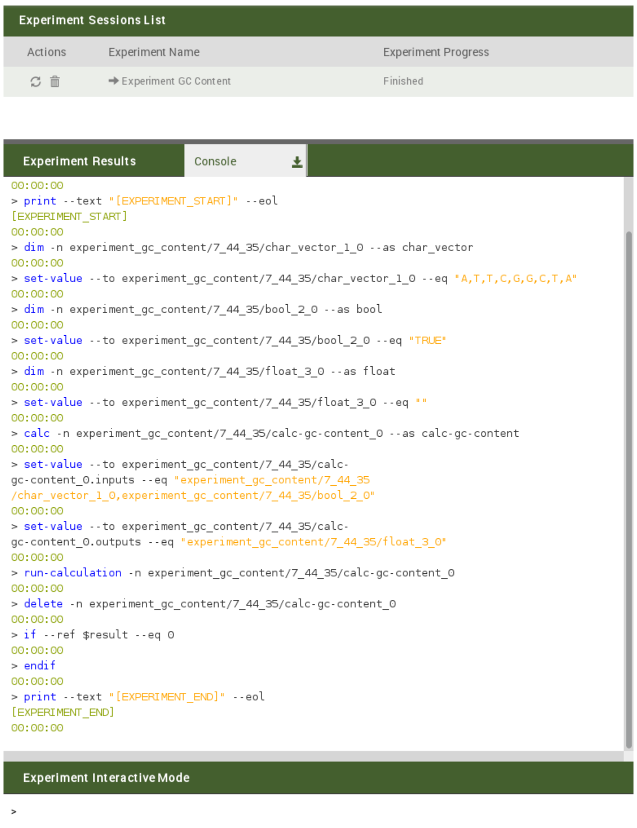
We are going to execute the experiment. Please, see Figure 10.1b

- Click on the *Play* button. You will see your experiment on the *Experiment Sessions List*.
- If the experiment is Finished, press it with the mouse. You will see all the executed instructions on the Console window.
- In the *Experiment Interactive Mode* window write: (use the correct output variable name)

```
print -n experiment_gc_content/14_47_5/float_3_0
You will see the experiment result:
------------------------------------------------------------------
 Attribute                 Type                      Value
------------------------------------------------------------------
 value                     float                     1.25
------------------------------------------------------------------
```



(a) Experiment's Graph



```
00:00:00
> print --text "[EXPERIMENT_START]" --eol
[EXPERIMENT_START]
00:00:00
> dim -n experiment_gc_content/7_44_35/char_vector_1_0 --as char_vector
00:00:00
> set-value --to experiment_gc_content/7_44_35/char_vector_1_0 --eq "A,T,T,C,G,G,C,T,A"
00:00:00
> dim -n experiment_gc_content/7_44_35/bool_2_0 --as bool
00:00:00
> set-value --to experiment_gc_content/7_44_35/bool_2_0 --eq "TRUE"
00:00:00
> dim -n experiment_gc_content/7_44_35/float_3_0 --as float
00:00:00
> set-value --to experiment_gc_content/7_44_35/float_3_0 --eq ""
00:00:00
> calc -n experiment_gc_content/7_44_35/calc-gc-content_0 --as calc-gc-content
00:00:00
> set-value --to experiment_gc_content/7_44_35/calc-
gc-content_0.inputs --eq "experiment_gc_content/7_44_35
/char_vector_1_0,experiment_gc_content/7_44_35/bool_2_0"
00:00:00
> set-value --to experiment_gc_content/7_44_35/calc-
gc-content_0.outputs --eq "experiment_gc_content/7_44_35/float_3_0"
00:00:00
> run-calculation -n experiment_gc_content/7_44_35/calc-gc-content_0
00:00:00
> delete -n experiment_gc_content/7_44_35/calc-gc-content_0
00:00:00
> if --ref $result --eq 0
00:00:00
> endif
00:00:00
> print --text "[EXPERIMENT_END]" --eol
[EXPERIMENT_END]
00:00:00
```

```
>
```

(b) Experiment's Command Line

Figure 10.1: Experiment Graph & Command Line

Part III

Continuous Integration

Chapter 11

Project Teamwork Methodology

After reading this chapter you should be able to collaborate with the project team, realize changes to the source code, test them and create a new release.

11.1 System Users

These are the main system users:

- **Software Engineer**: It is the developer that creates the system and the continuous integration environment
- **Pipeline Developer**: It is the user who develops new analysis pipelines combining the existing available calculations
- **Calculation Developer**: It is the software developer that uses the system to create new calculations
- **End User**: It is the user who analyzes his/her data using already available pipelines
- **Web Developer**: It is the developer that creates and maintains the project website

11.2 Vincent Driessen's branching model

Follow the following steps for modifying the any file of the project:
Please refer Figure 11.1 for Vincent Driessen's branching model.

- Download the project files to your computer:

```
$ git clone https://gitlab.bio.crag.local/gvera/ngasp.git
```

- Your current branch is the *master*. Change to the *develop* one:

```
$ git checkout develop
```

- If you had already the *develop* branch in your computer for some time, be sure that it has not modified by another developer in that time. Run a pull action for getting remote changes if they exist:

```
$ git pull
```

- You will create your own branch. For example, *feature-calc-gccontent*:

```
$ git branch feature-calc-gccontent
$ git checkout feature-calc-gccontent
```

- Now, you can modify the project files for adding the new feature. And when you finish it, add your changes to your local repository:

```
$ git add . --all
$ git commit -am "I added a new calculation to the project called GCContent."
```

- Your feature is not finished yet but you want to upload all your changes to the remote repository (for backup purposes, ...). Run:

```
$ # git pull <---- Before a push run always a pull if you share this branch with another
    developer. Maybe his/her changes break your code... So, if it happens maybe you don't
    want to upload your code, yet.
$ git push # <-- Upload your code to the remote repository
```

- You finished. It is time to upload your files to the remote repository. Do it in the same way as described before:

```
$ git pull
$ git push
```

- Now, you can merge your branch with the *develop* branch. Do:

```
$ git checkout develop                    # switch to the develop branch
$ git merge feature-calc-gccontent        # do the merge
```

- Once, it is decided that the *develop* branch can be a release, you can merge the *develop* branch with the *master* branch:

```
$ git checkout master                     # switch to the master branch
$ git merge develop                       # do the merge
```

- If Git shows conflicts that can no be resolved automatically. Do:

```
$ sudo yum install meld                   # install meld
$ git config --global merge.tool meld     # Tell git that this is your merge tool
$ git mergetool                           # Resolve conflicts manually
$ git add .                               # Confirm
$ git commit -am "Conflicts resolved manually."
```

11.3 Pull Request

This project is intended to be stored into the GitHutb repository. Once it is there, external contributor developers will be able to create a Fork of this project, update it and ask for a pull request to the project's owner.

See [Pull Request] for further information about Pull Requests.

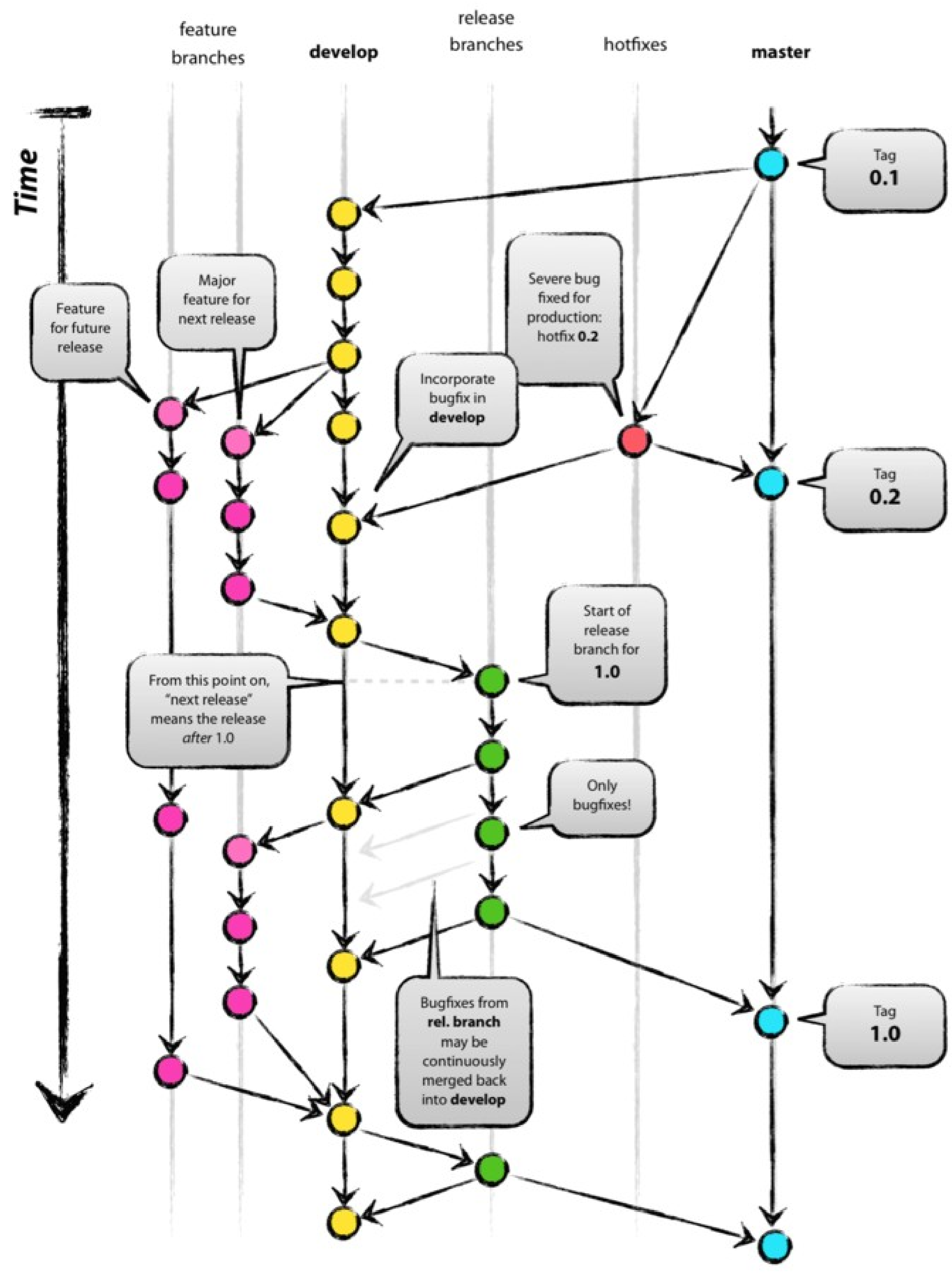

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Severe bug fixed for production: hotfix 0.2
Major feature for next release
Feature for future release
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Tag 1.0
Bugfixes from rel. branch may be continuously merged back into develop

Figure 11.1: Vincent Driessen's Branching Model

Chapter 12

Continuous Integration Environment

After reading this chapter you should be able to understand the Continuous Integration Environment architecture and to configure and run jobs.

12.1 Continuous Integration Environment Architecture

The continuous integration environment is responsible for compiling the project source code (back-end and all its external tools) and for publishing the release on the public web site.

This environment has one computer with Jenkins installed on it and some computers or virtual machines (called nodes) that do the job: Please see Figure 12.1 for an example calculations.

- **`ci-build-jenkins`** (`10.200.9.60`)
  This computer has installed Jenkins on it and it manages all computers.
  Access it via ssh. Run:

```
ssh -X ci_jenkins@ci-build-jenkins
$ firefox &
```

- **`ci-build-lin`** (`10.200.9.61`):
  It is a Jenkins' node.
  A Centos 7 virtual machine.
  Access it via ssh. Run:

```
$ ssh ci_ngasp@ci-build-lin
```

The following Jenkins' Nodes are not longer used since the environment is created with Docker:

- **`ci-build-win`** (`10.200.9.62`):
  It is a Jenkins' node.
  A Windows virtual machine.
  Access it via ssh. Run:

```
$ ssh ci_ngasp@ci-build-win
```

- **`ci-build-mac`** (`10.200.9.63`):
  It is a Jenkins' node.
  An OS X computer.
  Access it via VCN `http://localhost:60000`
  Use the Cluster for creating a tunnel. On the Cluster, run:

```
ssh -f -N -X -L localhost:60000:localhost:5900 ci_ngasp@10.200.9.63
```

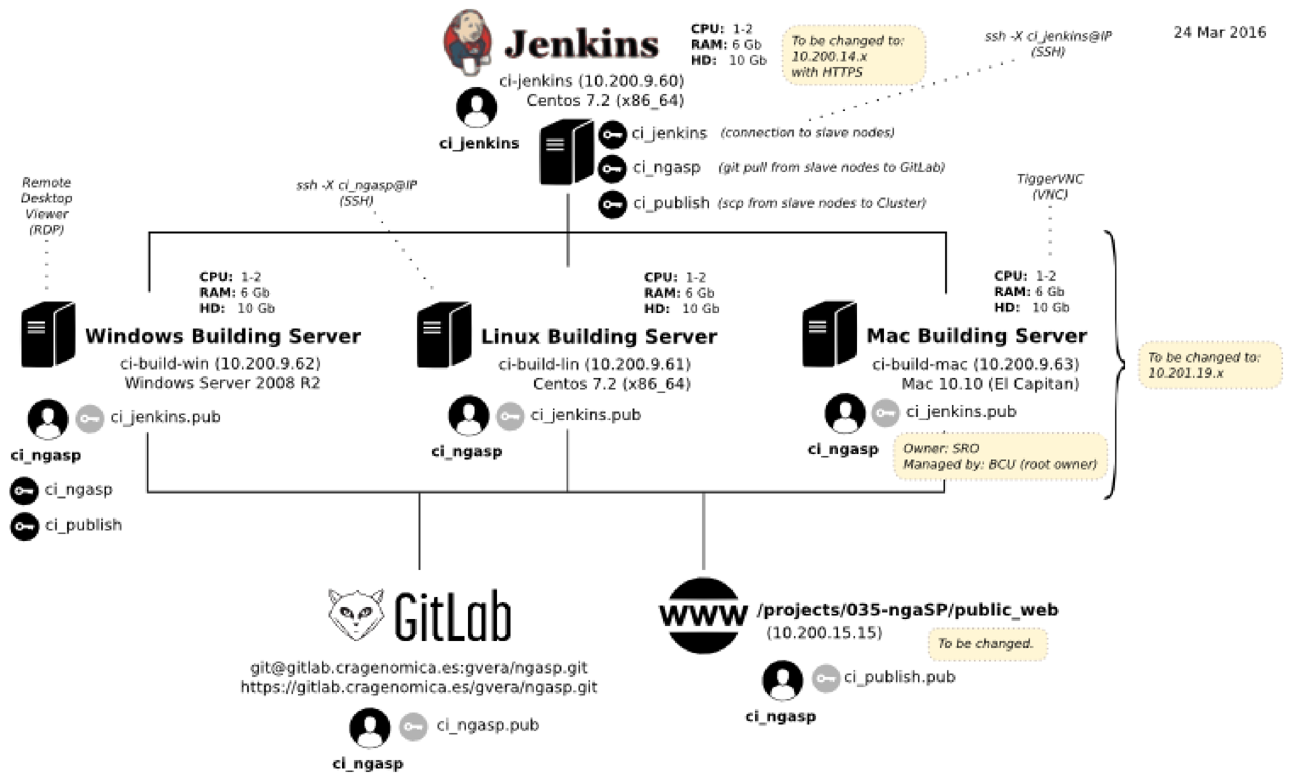

Jenkins
ci-jenkins (10.200.9.60)
Centos 7.2 (x86_64)
CPU: 1-2
RAM: 6 Gb
HD: 10 Gb
To be changed to: 10.200.14.x with HTTPS
ssh -X ci_jenkins@IP (SSH)
24 Mar 2016
ci_jenkins
ci_jenkins (connection to slave nodes)
ci_ngasp (git pull from slave nodes to GitLab)
ci_publish (scp from slave nodes to Cluster)
Remote Desktop Viewer (RDP)
ssh -X ci_ngasp@IP (SSH)
TiggerVNC (VNC)
Windows Building Server
ci-build-win (10.200.9.62)
Windows Server 2008 R2
Linux Building Server
ci-build-lin (10.200.9.61)
Centos 7.2 (x86_64)
Mac Building Server
ci-build-mac (10.200.9.63)
Mac 10.10 (El Capitan)
To be changed to: 10.201.19.x
ci_jenkins.pub
ci_ngasp
ci_publish
Owner: SRO
Managed by: BCU (root owner)
GitLab
git@gitlab.cragenomica.es:gvera/ngasp.git
https://gitlab.cragenomica.es/gvera/ngasp.git
ci_ngasp.pub
WWW /projects/035-ngaSP/public_web
(10.200.15.15)
To be changed.
ci_publish.pub

Figure 12.1: Continuous Integration Overview

On your workstation, run:

```
ssh -f -N -X -L localhost:60000:localhost:60000 jjene@10.201.19.100
```

12.2 Jenkins Jobs

Jenkins is able to perform some tasks periodically. Let's see how Jenkins creates a new downloadable package every night.

First of all, lets access Jenkins:

If you are at CRAG, third floor, do:

```
$ ssh -X jjene@10.201.19.100        # Login the Cluster
$ ssh -X ci_jenkins@10.200.9.60     # Login the Jenkins
$ firefox &                         # Open http://localhost:8080
```

The Jenkins homepage shows a list of jobs. Every job runs a set of tasks. The job name shows the type of tasks that it performs. If the job name contains:

- **lin**: The job is executed in the Linux node.
- **win**: The job is executed in the Windows node. Since we use Docker, it is not need to use this node.
- **mac**: The job is executed in the Mac OS X node. Since we use Docker, it is not need to use this node.
- **all**: The job is executed in all nodes.
- **clone**: The job downloads source code from the repository to the node. If you execute this task, first you must delete the `ngasp` folder manually from the node.

- **pull**: The job uploads source code to the repository. It is used when you change the source code in the node and you want to upload the code to the repository. You can not do a "git pull" from the node but you can do it with this job.
- **job**: The job downloads the source code, it compiles it and it sends the release to the public web site.

Most of the tasks are called from Jenkins but are defined in Gradle configuration files. Next section will describe how.

12.3 Jenkins and Gradle

Most of Jenkins tasks are defined in Gradle configuration files. This, allows the user to execute them from a terminal window, also. These configuration files are stored here:
`<ngasp_folder>/develop/.gradle/` and these are the main ones:

- **`deploy_ngasp.gradle`**: Here, all tasks are defined.
- **`deploy_ngasp_linux.gradle`**: Here, platform configuration is defined (paths, binary names, compilers, ssh tools, ...).
- **`build_backend.gradle`**: This is the main back-end compilation configuration file.
- **`build_backend_linux.gradle`**: Here is the right configuration for compiling the back-end source code in Linux (compiler and linker options).

Most of Gradle configuration files have one more configuration files per platform (`_linux.gradle`, `_win.gradle` and `_mac.gradle`). But, since Docker is used, only configuration files ended with `_linux.gradle` are used.

The main job is named *ngasp_all_build*. Lets see what does it do.

12.4 ngasp_all_build

This job downloads all source codes, it compiles them and it sends the created release to the public web site.

In particular, it does the following:

1. Specify where it has to be executed: ci-build-lin.
2. Specify when it has to be executed: Every day at 7 o'clock.
3. Specify the SSH Agent for getting the source code: *ci_ngasp* user with *ci_ngasp* key.
4. Specify the SSH Agent for sending the package to the website: *ci_ngasp* user with *ci_publish* key.
5. Run *getCodeFromRepositoryIgnoreLocal* task: This task downloads the last version of the *release* branch from the project repository .
6. Run *unpackExamples* task: This task uncompresses some big example data files that were compressed inside the repository due to their huge size.
7. Run *getLastProjectVersion* task: This task gets the project version from Git and generates a C++ file with a version constant prepared to be compiled with all the project source code.
8. Run *buildDockers* task: This task creates the compilation environment.
9. Run *compileSourceCode* task: This task compiles all source codes using the compilation environment. It calls the `compile_all.sh` script. For compiling the back-end source code the script uses Gradle with `build_backend.gradle`.

10. Run *copyBinaries* task: This task copies all created binaries to the release folder.
11. Run *compressRelease* task: This task creates a compressed ZIP package using the `release` folder.
12. Run *publishLastRelease* task: This task sends the package to the web site.

Note: Jenkins builds are stored in the `ci-jenkins` machine in this folder: `/var/lib/jenkins/jobs`

12.5 Adding a new node to Jenkins

This section describes how to add a virtual machine as a new Jenkins node.

1. First of all, create a new virtual machine. For example:

```
Operating System:       Centos 7
Architecture:           x86_64
CPU op-mode(s):         32-bit, 64-bit
CPU(s):                 4
Model name:             Intel(R) Xeon(R) CPU           X5550  @ 2.67GHz
KiB Mem :               5.946.436 KiB
KiB Swap:               524.284 KiB
HD:                     31G
```

2. Create the user ci_ngasp and add it to the sudoers group:

```
useradd ci_ngasp
passwd ci_ngasp
sudo usermod -a -G wheel ci_ngasp
```

3. Logout and login with this user:

```
exit
ssh ci_ngasp@10.200.9.61
```

4. Create the workspace folder for Jenkins in this server. Then, assign user privileges to the folder. Maybe you will need to repeat the privileges assignation when subfolders are created.

```
sudo mkdir /var/jenkins
sudo mkdir /var/jenkins/workspace
sudo mkdir /var/jenkins/workspace/ngasp_build_linux
sudo chmod 777 -R /var/jenkins
```

5. Ensure that the computer has always the right date and time installing ntpdate and enblabling the service at boot time:

```
# The installation folder is be /usr/sbin/ntpdate
sudo yum install ntp ntpdate ntp-doc
chkconfig ntpd on
sudo ntpdate pool.ntp.org
ntpdate -q ntp.uab.es
sudo ntpdate ntp.uab.es
sudo vi /etc/ntp.conf
server ntp.uab.es
systemctl enable ntpdate.service
# Restart the service
# systemctl restart ntpd.service
systemctl enable ntpd.service # enable it at boot time
```

6. Check if ntp is running:

```
ps aux | grep ntp
# Can you see it? Then it is Ok.
```

7. Install some tools for installing next packages:

```
sudo yum install zip
sudo yum install unzip
sudo yum install wget
```

8. Install Java JDK (Gradle requirement):

```
sudo yum install java-1.8.0-openjdk
```

9. Install Gradle:

```
wget https://services.gradle.org/distributions/gradle-3.1-bin.zip
unzip ./gradle-3.1-bin.zip -d /home/ci_ngasp
rm ./gradle-3.1-bin.zip
sudo vi /etc/profile.d/gradle.sh   # in this file if you access from interactive login
sudo vi ~/.bashrc                  # in this file if you access from a console in graphical
     mode
export PATH=$PATH:/home/ci_ngasp/gradle-3.1/bin
# Use the Gradle Daemon
mkdir /home/ci_ngasp/.gradle
sudo vi /home/ci_ngasp/.gradle/gradle.properties
org.gradle.daemon=true
```

10. Install Docker:

```
sudo yum update
tee /etc/yum.repos.d/docker.repo <<-'EOF'
[dockerrepo]
name=Docker Repository
baseurl=https://yum.dockerproject.org/repo/main/centos/7/
enabled=1
gpgcheck=1
gpgkey=https://yum.dockerproject.org/gpg
EOF
sudo yum install docker # or docker-engine
sudo systemctl enable docker.service
sudo systemctl start docker
sudo groupadd docker
sudo usermod -aG docker ci_ngasp
sudo newgrp docker
```

11. Install Docker Compose:

```
curl -L https://github.com/docker/compose/releases/download/1.9.0-rc1/docker-compose-`uname
     -s`-`uname -m` > /usr/bin/docker-compose
chmod +x /usr/bin/docker-compose
```

12. Prepare Docker for Jenkins:

```
sudo vi /usr/lib/systemd/system/docker.service
ExecStart=/usr/bin/docker-current daemon \
      -H unix:// -H tcp://localhost:2375 \
```

13. Create a file where all environment constants will be placed. This file will be loaded by Jenkins at the beginning of the build process:

```
# Run this command and locate something like this "tcp://localhost:2375":
docker-compose --version
# Create a file named and add the previous line:
vi /home/ci_ngasp/environment
DOCKER_HOST=tcp://localhost:2375
```

14. Install Git:

```
sudo yum install git
```

15. Add Keys to the machine:

```
# From ci-build-lin
mkdir /home/ci_ngasp/.ssh
# From the workstation
scp ./keys/ci_jenkins-id_rsa.pub jjene@10.201.19.100:/home/jjene/ci_jenkins-id_rsa.pub
# From the cluster
ssh-copy-id -i ./ci_jenkins-id_rsa.pub ci_ngasp@10.200.9.61
# From Jenkins node maybe you have to update or remove /home/ci_jenkins/.ssh/known_hosts
# From ci-build-lin. Important!
chmod 700 /home/ci_ngasp/.ssh
chmod 640 .ssh/authorized_keys
```

16. Create the Gradle wrapper:

```
cd /home/ci_ngasp
gradle wrapper --gradle-version 3.1
cd /var/jenkins/workspace/ngasp_lin_build
```

17. Do a first clone attempt of the project. The objective is to answer 'yes' to a question and then stop:

```
cd $user_home
git clone -b develop git@gitlab.bio.crag.local:gvera/ngasp.git
answer 'yes' to the authenticity of host question.
```

18. Do a first access attempt to ci-publish from ci-build-lin. Same objective as before: answer 'yes' to a question and stop:

```
ssh ci_ngasp@10.200.15.15
The authenticity of host '10.200.15.15 (10.200.15.15)' can't be established.
RSA key fingerprint is ab:f4:c5:e5:ca:77:ab:21:6d:5b:51:73:08:83:03:21.
Are you sure you want to continue connecting (yes/no)? yes
```

19. Reassign user privileges to the Jenkins workspace folder:

```
chmod 777 -R /var/jenkins
```

20. Execute this to enable Docker access to the host volumes:

```
su -c "setenforce 0"
```

12.6 Build and Publish Now

If you do not want to wait for Jenkins to publish the last release of the Git **develop branch**, you can do it manually, running:

```
$ sh -X jjene@10.201.19.100
[jjene@node028 ~]$ ssh -X ci_jenkins@10.200.9.60
[ci_jenkins@ci-jenkins ~]$ firefox &
```

Then, do `ngasp_lin_build > Build now`.

When finished, the release will appear at `https://bioinformatics.cragenomica.es/projects/ngaSP/downloads/index.php`

Part IV

Calculation Development

Chapter 13

Creating a New Calculation

After reading this chapter you should be able to create a new calculation inside the ngasp system. Please see Figure 13.1 for an example calculation.

Before modifying the system you must follow the Teamwork Methodology (See Section 11) and create a new branch to work on it. For morre information about the module of the next example see Section 3.1.

13.1 Creation of the *GC Content* Calculation

In molecular biology and genetics, *GC Content* (or guanine-cytosine content) is the percentage of nitrogenous bases on a DNA molecule that are either guanine or cytosine (from a possibility of four different ones, also including adenine and thymine).

GC Content is usually expressed as a percentage value, but sometimes as a ratio (called G+C ratio or GC-ratio). *GC Content* percentage is calculated as Formula 13.1

$$percentage = \frac{G+C}{A+T+G+C} \tag{13.1}$$

whereas the AT/GC ratio is calculated as Formula 13.2

$$ratio = \frac{A+T}{G+C} \tag{13.2}$$

1. Create three constants for your calculation: the calculation's name and calculation's descriptions:

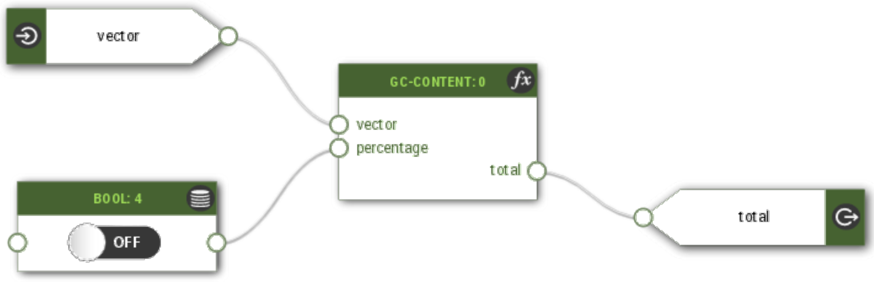


Figure 13.1: *GC Content* Calculation

Add the highlighted code in the right position replacing the text code by your own text code. For example, inside the KeyString enum and just before the _CALC_LAST constant.

ngaSP/Source Files/ngaSP/language/CStringTable.h

```
enum KeyString {
  ...
  CALC_GCCONTENT,
  CALC_GCCONTENT_BRIEF_DESC,
  CALC_GCCONTENT_DESC,

  _CALC_LAST,
  ...
}
```

2. Write your calculation name and description:
ngaSP/Source Files/ngaSP/language/CStringTable.cpp

```
CStringTable::CStringTable() {
  ...
  strings_[ENGLISH_COL][CALC_GCCONTENT]            = "gc-content";
  strings_[ENGLISH_COL][CALC_GCCONTENT_BRIEF_DESC] = "";
  strings_[ENGLISH_COL][CALC_GCCONTENT_DESC]  = "Calc description";

  strings_[ENGLISH_COL][_CALC_LAST]              = "";
  ...
}
```

3. Add your calculation to the system:
ngaSP/Source Files/ngaSP/calculations/CCalcFactory.cpp

```
...
#include "Calc/CCalcGCContent.h"

CCalcFactory::CCalcFactory() {
  ...
  data_map_[KeyString::CALC_GCCONTENT] = new CCalcGCContent();
}
```

4. Create a class for your calculation. Use these patterns for the ".h" & ".cpp" files:
ngaSP/Source Files/ngaSP/calculations/Calc/CCalcGCContent.h

```
/**
 *   \brief     CCalcGCContent.h
 *   \details
 *   \author
 *   \version
 *   \date
 */

#ifndef CALCULATIONS_CALCS_CCALGCCONTENT_H_
#define CALCULATIONS_CALCS_CCALGCCONTENT_H_

#include "../ICalculation.h"

class CCalcGCContent : public ICalculation {
 public:
  CCalcGCContent();
  virtual ~CCalcGCContent();

 public:
  void Prepare(void);
  void Calculate(bool dry_run);
  void Finalize(void);

 public:
  ICalculation* clone() const {
     return new CCalcGCContent();
  }
};

#endif  // CALCULATIONS_CALCS_CCALGCCONTENT_H_
```

ngaSP/Source Files/ngaSP/calculations/Calc/CCalcGCContent.cpp

```
/**
 *  \brief     CCalcGCContent.cpp
 *  \details
 *  \author
 *  \version
 *  \date
 */

#include "CCalcGCContent.h"

#include "../../language/CStringTable.h"

CCalcGCContent::CCalcGCContent()
: ICalculation(KeyString::CALC_GCCONTENT,
               KeyString::CALC_GCCONTENT_BRIEF_DESC,
               KeyString::CALC_GCCONTENT_DESC,
               KeyString::NGASP_AUTHORS,
               KeyString::MSTATSPOP_COPYRIGHT,
               KeyString::GENERIC_CITATIONS,
               KeyString::UNDEFINED_STRING) {
}

CCalcGCContent::~CCalcGCContent() {
}

void CCalcGCContent::Prepare(void) {
}

void CCalcGCContent::Calculate(bool dry_run) {
  if (dry_run == true) {
    return;
  }

}

void CCalcGCContent::Finalize(void) {
}
```

5. Define your calculation's inputs and outputs:
ngaSP/Source Files/ngaSP/calculations/Calc/CCalcGCContent.h

```
/**
 *  \brief     CCalcGCContent.h
 *  \details
 *  \author
 *  \version
 *  \date
 */

#ifndef CALCULATIONS_CALCS_CCALGCCONTENT_H_
#define CALCULATIONS_CALCS_CCALGCCONTENT_H_

#include "../ICalculation.h"

// Include the data types you use:
#include "../../data_manager/CDataManager.h"
#include "../../data_manager/Data/CDataCharVector.h"
#include "../../data_manager/Data/CDataBoolean.h"
#include "../../data_manager/Data/CDataFloat.h"


class CCalcGCContent : public ICalculation {
 public:
  CCalcGCContent();
  virtual ~CCalcGCContent();

 public:
  void Prepare(void);
  void Calculate(bool dry_run);
  void Finalize(void);

 private:
  // Declare your inputs and outputs:
  // Inputs
  CDataCharVector *vector;
  CDataBoolean *percentage;

  // Outputs
  CDataFloat *total;

 public:
  ICalculation* clone() const {
     return new CCalcGCContent();
  }
};

#endif  // CALCULATIONS_CALCS_CCALGCCONTENT_H_
```

ngaSP/Source Files/ngaSP/calculations/Calc/CCalcGCContent.cpp

```
/**
 *  \brief     CCalcGCContent.cpp
 *  \details
 *  \author
 *  \version
 *  \date
 */

...

CCalcGCContent::CCalcGCContent()
: ICalculation(KeyString::CALC_GCCONTENT,
               KeyString::CALC_GCCONTENT_BRIEF_DESC,
               KeyString::CALC_GCCONTENT_DESC,
               KeyString::NGASP_AUTHORS,
               KeyString::MSTATSPOP_COPYRIGHT,
               KeyString::GENERIC_CITATIONS,
               KeyString::UNDEFINED_STRING) {
```

```
  // Define here the inputs and outputs of your calculation:

  BEGIN_CALCULATION_INTERFACE_DEFINITION
    SET_INPUT_INFO(vector,                           // Variable
                   UNDEFINED_STRING,                 // Group
                   CALC_GCCONTENT_VECTOR,            // Short Name
                   CALC_GCCONTENT_VECTOR_LONG,       // Long Name
                   CALC_GCCONTENT_VECTOR_DESC,       // Description
                   CALC_GCCONTENT_VECTOR_SAMP,       // Example
                   CALC_GCCONTENT_VECTOR_ONLY,       // Use only if
                   CALC_GCCONTENT_VECTOR_DEFV,       // Default value
                   UNDEFINED_VALUE,                  // Min. Value
                   UNDEFINED_VALUE,                  // Max. Value
                   OPTTYPE_mandatory)                // Required)
    SET_INPUT_INFO(percentage,                       // Variable
                   UNDEFINED_STRING,                 // Group
                   CALC_GCCONTENT_PERCENTAGE,        // Short Name
                   CALC_GCCONTENT_PERCENTAGE_LONG,   // Long Name
                   CALC_GCCONTENT_PERCENTAGE_DESC,   // Description
                   CALC_GCCONTENT_PERCENTAGE_SAMP,   // Example
                   CALC_GCCONTENT_PERCENTAGE_ONLY,   // Use only if
                   CALC_GCCONTENT_PERCENTAGE_DEFV,   // Default value
                   UNDEFINED_VALUE,                  // Min. Value
                   UNDEFINED_VALUE,                  // Max. Value
                   OPTTYPE_mandatory)                // Required)

    SET_OUTPUT_INFO(total,                           // Variable
                   UNDEFINED_STRING,                 // Group
                   CALC_GCCONTENT_TOTAL,             // Short Name
                   CALC_GCCONTENT_TOTAL_LONG,        // Long Name
                   CALC_GCCONTENT_TOTAL_DESC,        // Description
                   CALC_GCCONTENT_TOTAL_SAMP,        // Example
                   CALC_GCCONTENT_TOTAL_ONLY,        // Use only if
                   CALC_GCCONTENT_TOTAL_DEFV,        // Default value
                   UNDEFINED_VALUE,                  // Min. Value
                   UNDEFINED_VALUE,                  // Max. Value
                   OPTTYPE_mandatory)                // Required)
  END_CALCULATION_INTERFACE_DEFINITION
}

CCalcGCContent::~CCalcGCContent() {
}

void CCalcGCContent::Prepare(void) {
  // Get your inputs and outputs:
  DM_GET_INPUTS
    DM_INPUT(vector)
    DM_INPUT(percentage)
  DM_GET_OUTPUTS
    DM_OUTPUT(total)
  DM_END
}

void CCalcGCContent::Calculate(bool dry_run) {
  if (dry_run == true) {
    return;
  }

}

void CCalcGCContent::Finalize(void) {
  DM_DEL_ALL_LOCAL_DATA
}
```

6. Write your calculation:
   ngaSP/Source Files/ngaSP/calculations/Calc/CCalcGCContent.cpp

```
/**
 *  \brief     CCalcGCContent.cpp
 *  \details
 *  \author
 *  \version
 *  \date
 */

...

void CCalcGCContent::Calculate(bool dry_run) {
  if (dry_run == true) {
    return;
  }

  // Calculate:

  total->set_value(0);

  int64_t T = 0;
  int64_t C = 0;
  int64_t G = 0;
  int64_t A = 0;

  for (int64_t i = 0; i < vector->Size(); i++) {
    switch(vector[i]) {
      case 'T':T++;break;
      case 'C':C++;break;
      case 'G':G++;break;
      case 'A':A++;break;
      default:break;
    }
  }

  int64_t numerator = 0;
  int64_t divisor = 0;

  if (percentage->value() == true) {
    numerator = (G+C);
    divisor = (A+T+G+C);
  } else {
    numerator = (A+T);
    divisor = (G+C);
  }

  if (divisor != 0) {
    float result = numerator;
    result /= divisor;
    total->set_value(result);
  }
}

...
```

7. Add the license & copyright note to your calculation:
   `ngaSP/Source Files/ngaSP/calculations/Calc/CCalcGCContent.h`

```
/* LICENSE & COPYRIGHT note */
/**
 *  \brief     CCalcGCContent.h
 *  \details
 *  \author
 *  \version
 *  \date
 */
```

   `CCalcGCContent.cpp`

```
/* LICENSE & COPYRIGHT note */
/**
 *  \brief     CCalcGCContent.cpp
 *  \details
 *  \author
 *  \version
 *  \date
 */
```

13.2 Testing the *GC Content* Calculation

Create the following ngasp script (`gc_content.ngasp`):

```
verbose --level debug

const --name FALSE --by 0
const --name TRUE  --by 1

dim -n sequence       --as char_vector
dim -n get_percentage --as bool
dim -n total          --as float

set-value --to sequence       --eq "A,T,T,C,G,G,C,T,A"
set-value --to get_percentage --eq TRUE

calc -n c1 --as calc-gc-content
set --to c1.inputs --eq "sequence,get_percentage"
set --to c1.outputs --eq "total"
run -n c1

print -n total --mode json --eol
exit
```

Run the ngasp script (`gc_content.ngasp`):

```
$ <bin_path>/ngasp load -i ./gc_content.ngasp

Verbose Level: debug
00:00:00
$ constant --name FALSE --by 0
00:00:00
$ constant --name TRUE --by 1
00:00:00
$ dim -n sequence --as char_vector
00:00:00
$ dim -n get_percentage --as bool
00:00:00
$ dim -n total --as float
00:00:00
$ set-value --to sequence --eq "A,T,T,C,G,G,C,T,A"
00:00:00
$ set-value --to get_percentage --eq 1
00:00:00
```

```
$ calc -n c1 --as calc-gc-content
00:00:00
$ set-value --to c1.inputs --eq "sequence,get_percentage"
00:00:00
$ set-value --to c1.outputs --eq "total"
00:00:00
$ run-calculation -n c1
00:00:00
$ print -n total --mode json --eol
{"id":"total","type":"float","value":"0.444444"}
00:00:00
$ exit
00:00:00
```

13.3 Advanced Calculation Development Commands

You can use the following macros in your calculations:

- *DM_RESULT*:

  The calculation always returns 0 unless you specify another return value. The OK value is 0. Other value indicates error. If the calculation in executed in a pipeline, the execution stops if it does not return 0.

```
void CCalc@GCContent@::Calculate(bool dry_run) {
  ...

  DM_RESULT(result)
  result->set_value(0);
}
```

- *DM_ITERATION_NUMBER* and *DM_ITERATION_VALUE*:

  If the calculation is executed in a pipeline you can access the iteration number and iteration value using these macros:

```
void CCalc@GCContent@::Calculate(bool dry_run) {
  ...
  DM_ITERATION_NUMBER(iteration_number)
  DM_ITERATION_VALUE(iteration_value)
}
```

  For example, you can use it for storing files that are created inside a loop:

```
void CCalc@GCContent@::Calculate(bool dry_run) {
  ...
  DM_ITERATION_NUMBER(iteration_number)
  DM_ITERATION_VALUE(iteration_value)

  output_file_name->set_value(
    CFile::ConcatenateIterationToFilePathName(
      output_file_name->value(),
      iteration_number->value(),
      iteration_value->value()));
}
```

  You will need to add the following includes for using these macros:

```
  #include "../../data_manager/Data/CDataInt.h"
  #include "../../data_manager/Data/CDataInt64.h"
  #include "../../data_manager/Data/CDataStdString.h"
  #include "../../util/CFile.h"
```

13.4 Messaging and Verbose Mode

Instead of using *printf* or *std::cout* functions, you should use the following macros for showing messages to the user. Doing it in this way the user will be able to filter the application output messages:

- Error Messages:

```
ERROR_MSG
  << "Incorrect number of bases: "
  << bases
  << ". Please do something."
END_MSG;
```

- Warning Messages:

```
WARNING_MSG
  << "Number of chromosomes lower than expected."
END_MSG;
```

- Debug Messages:

```
DEBUG_MSG
  << "x= " << x << EOL
  << "y= " << y << EOL
  << "z= " << z
END_MSG;
```

- Messages:

```
NORMAL_MSG
  << "** Calculation Results **" << EOL
  << "Number of bases: " << bases
END_MSG;
```

The messages that are shown to the user depend on the configured *verbose mode*. The verbose mode can be set using the command line. (Run *ngasp help -n verbose* for further information):

- *verbose –mode silent*: Only error messages will be shown to the user.
- *verbose –mode normal*: All messages will be shown to the user except the debug ones.
- *verbose –mode debug*: All messages will be shown to the user.

Regardless the verbose mode, all debug, error and warning messages will be stored in the `./ngasp.log` file. It is also stored the history of executed commands in the `./ngasp.history`,

Part V

Pipeline Development

Chapter 14

Experiment vs Pipeline

- *Experiment*: It is a workflow with real input / output data.

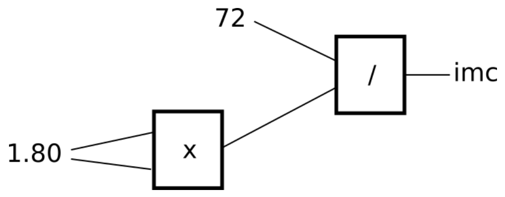


Figure 14.1: Experiment Example

- *Pipeline*: It is a generic workflow. It requires an experiment that executes it.

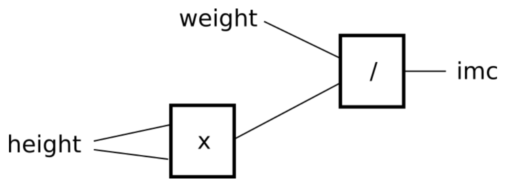


Figure 14.2: Pipeline Example

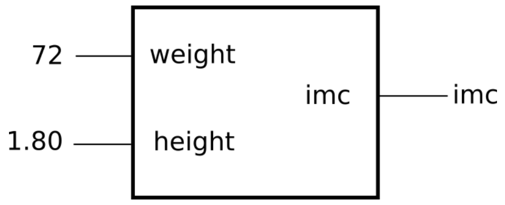


Figure 14.3: Pipeline Call Example

Note:
Another difference between *Experiments* and *Pipelines* is that pipelines can be looped, experiments cannot.

Chapter 15

Experiment Creation

- Objective: Get statistics from DNA sequences in fasta format:

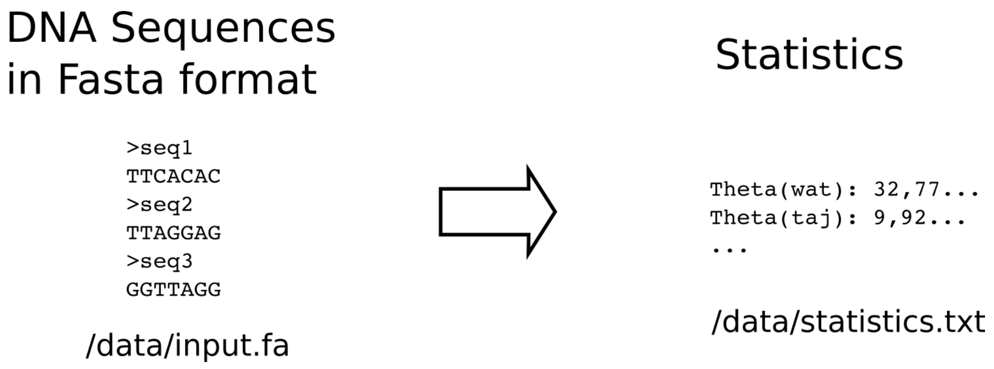


Figure 15.1: Experiment Creation - Objective

- Draft Workflow:

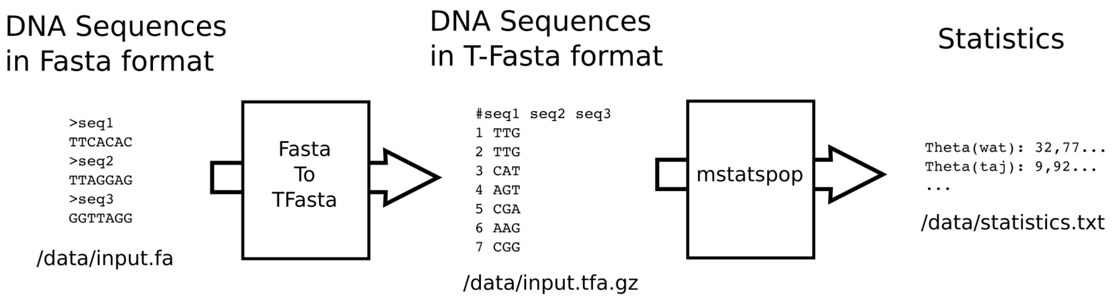


Figure 15.2: Experiment Creation - Draft Workflow

- ngasp Experiment: (See figure 20.1 for a real ngasp example)

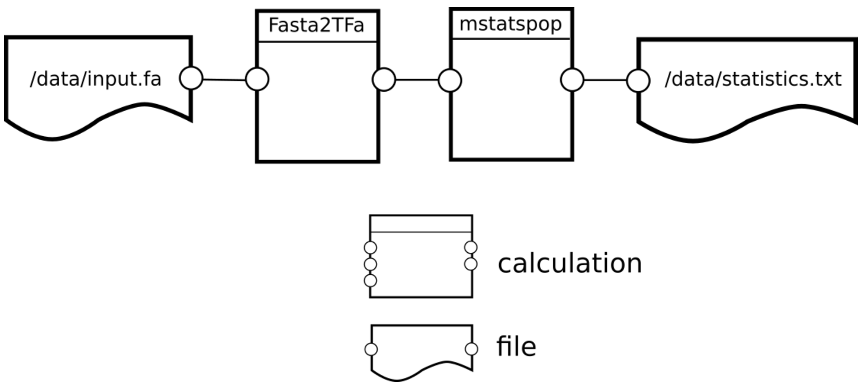


Figure 15.3: Experiment Creation - ngasp Experiment

Chapter 16

Creation of a Generic and Reusable Pipeline

- Desired New Pipeline Usage:

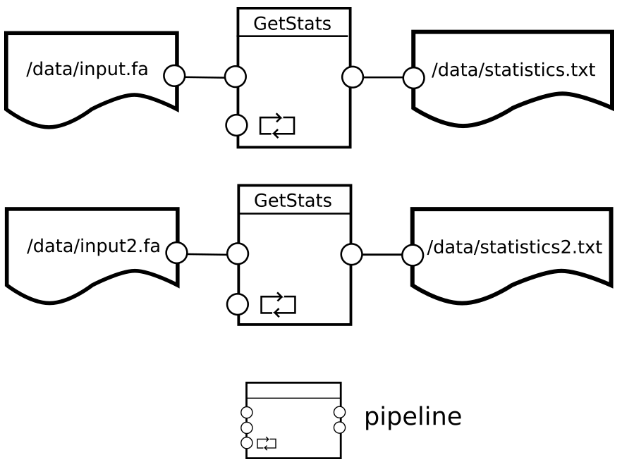


Figure 16.1: Pipeline Creation - Multiple Calls

- Create a Pipeline replacing input / otput data by pipeline inputs and outputs (See figure 20.2 for a real ngasp example):

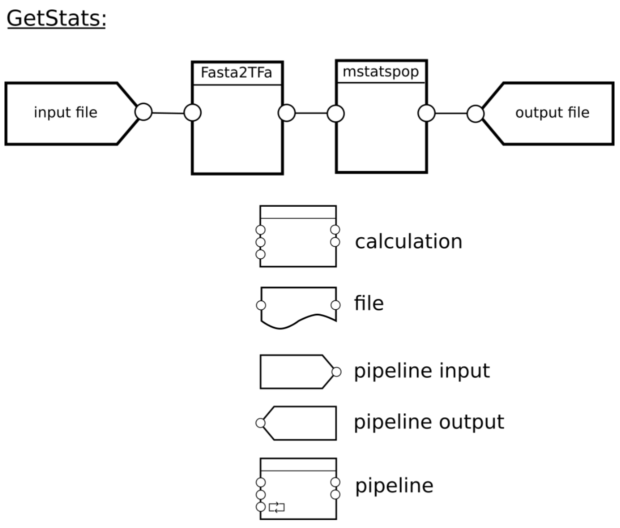


Figure 16.2: Pipeline Creation - ngasp Pipeline

Chapter 17

Pipeline Creation for Multiple Input and Output Files

- Objective:

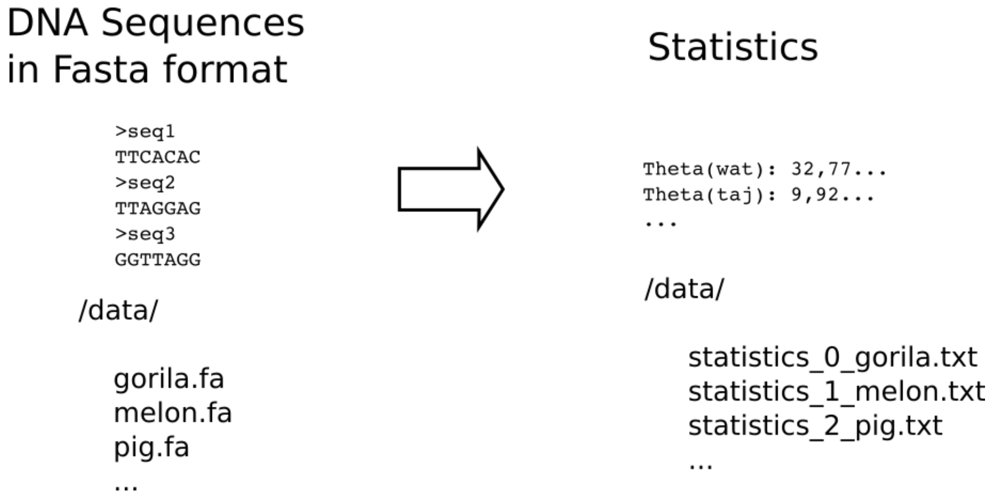


Figure 17.1: Pipeline Creation - Multiple Output Files

- Pipeline Desired Usage:

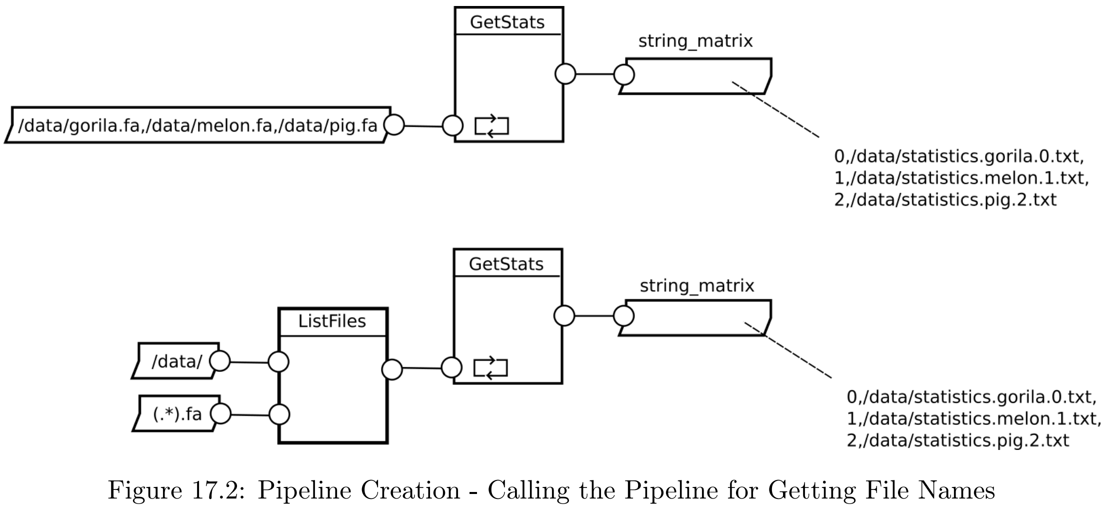

GetStats
string_matrix
/data/gorila.fa,/data/melon.fa,/data/pig.fa
0,/data/statistics.gorila.0.txt,
1,/data/statistics.melon.1.txt,
2,/data/statistics.pig.2.txt
GetStats
string_matrix
ListFiles
/data/
(.*).fa
0,/data/statistics.gorila.0.txt,
1,/data/statistics.melon.1.txt,
2,/data/statistics.pig.2.txt

Figure 17.2: Pipeline Creation - Calling the Pipeline for Getting File Names

- Pipeline Design (See figure 20.3 for a real ngasp example):

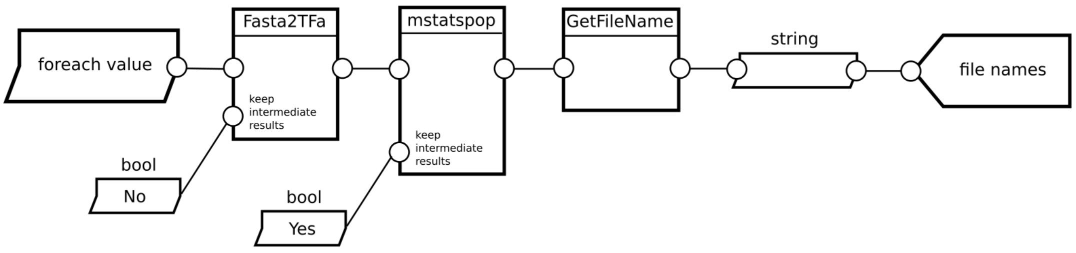


Figure 17.3: Pipeline Creation - Getting Multiple Output File Names

Chapter 18

Pipeline Creation for Multiple Input Files that Generate Only One Output File

- Objective:

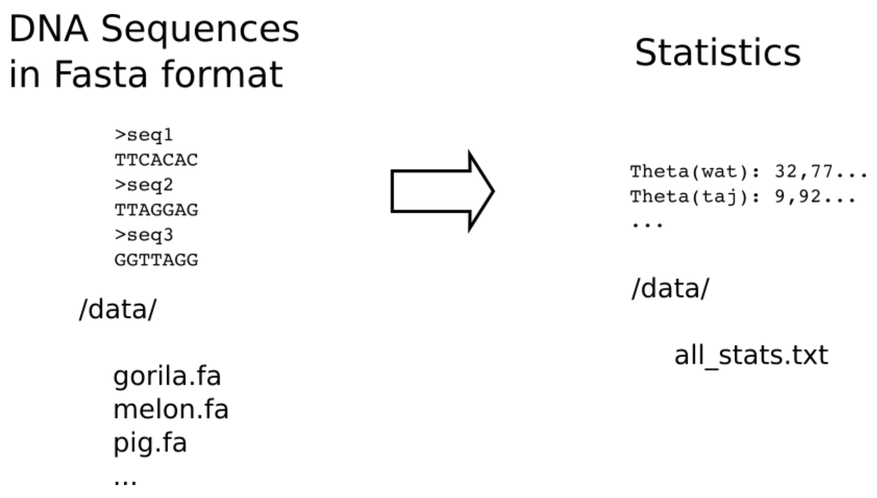


Figure 18.1: Pipeline Creation - One Output File Append

- Pipeline Desired Usage:

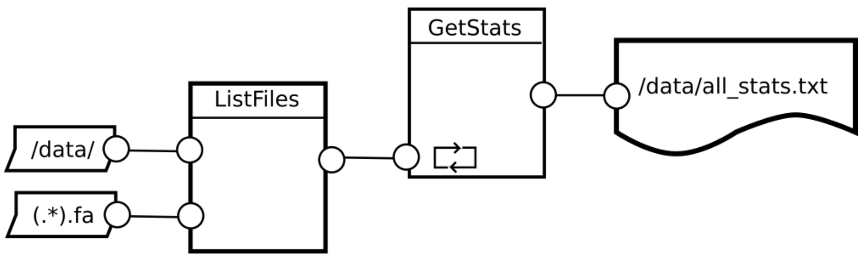


Figure 18.2: Pipeline Creation - Calling the Pipeline for Getting One Output File

- Pipeline Design (See figure 20.4 for a real ngasp example):

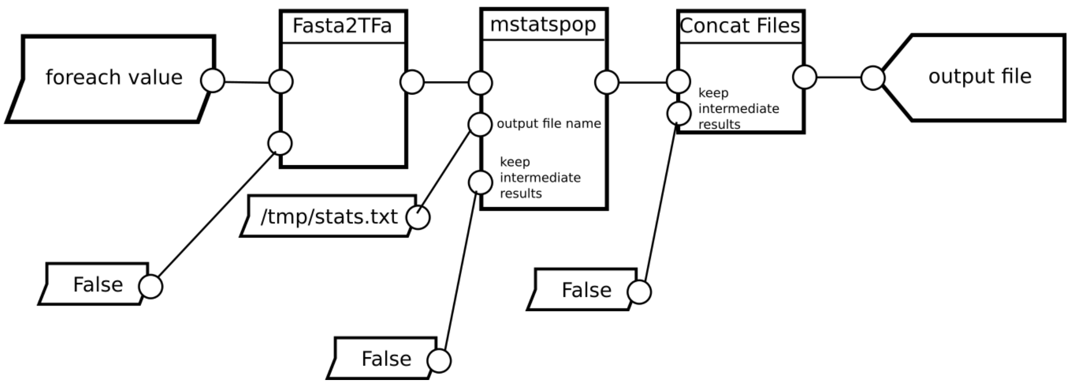


Figure 18.3: Pipeline Creation - Getting One Output File

Chapter 19

Pipeline Outputs Sumary

- Summary:

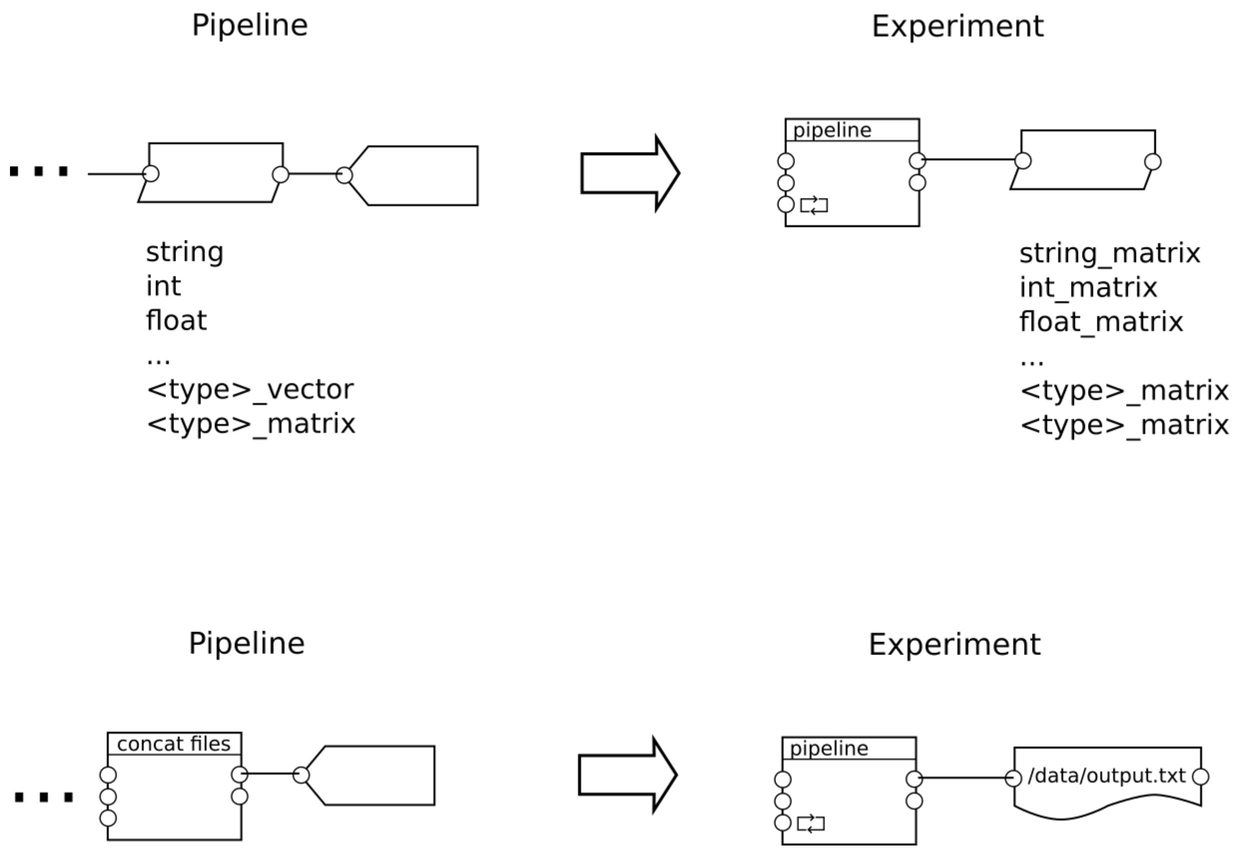


Figure 19.1: Pipeline Outputs Sumary

Chapter 20

ngasp Real Examples

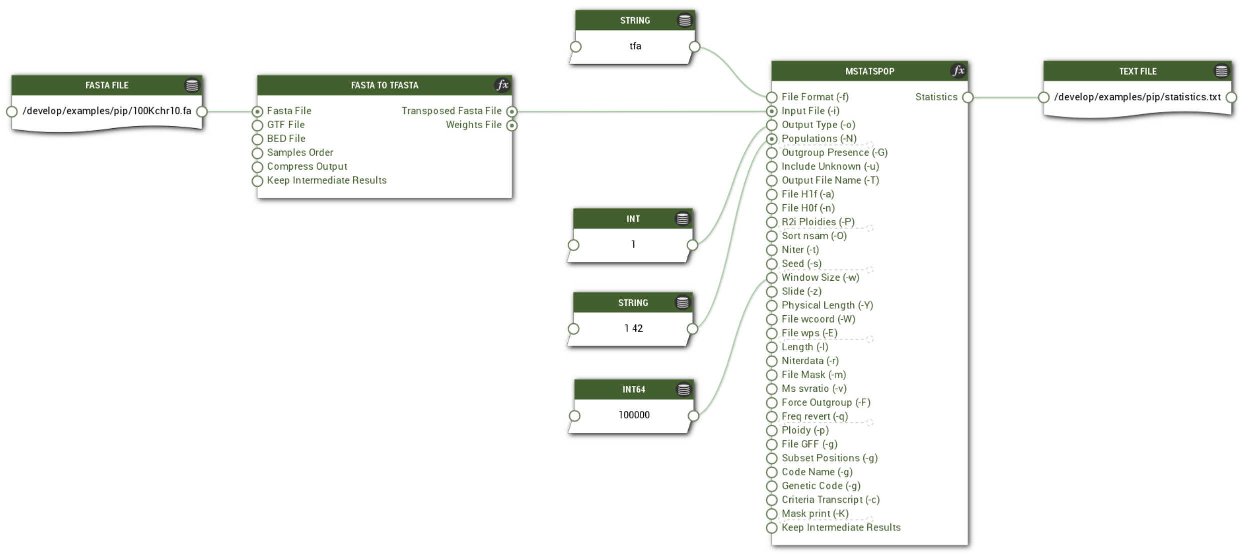


Figure 20.1: ngasp - Experiment Example 1

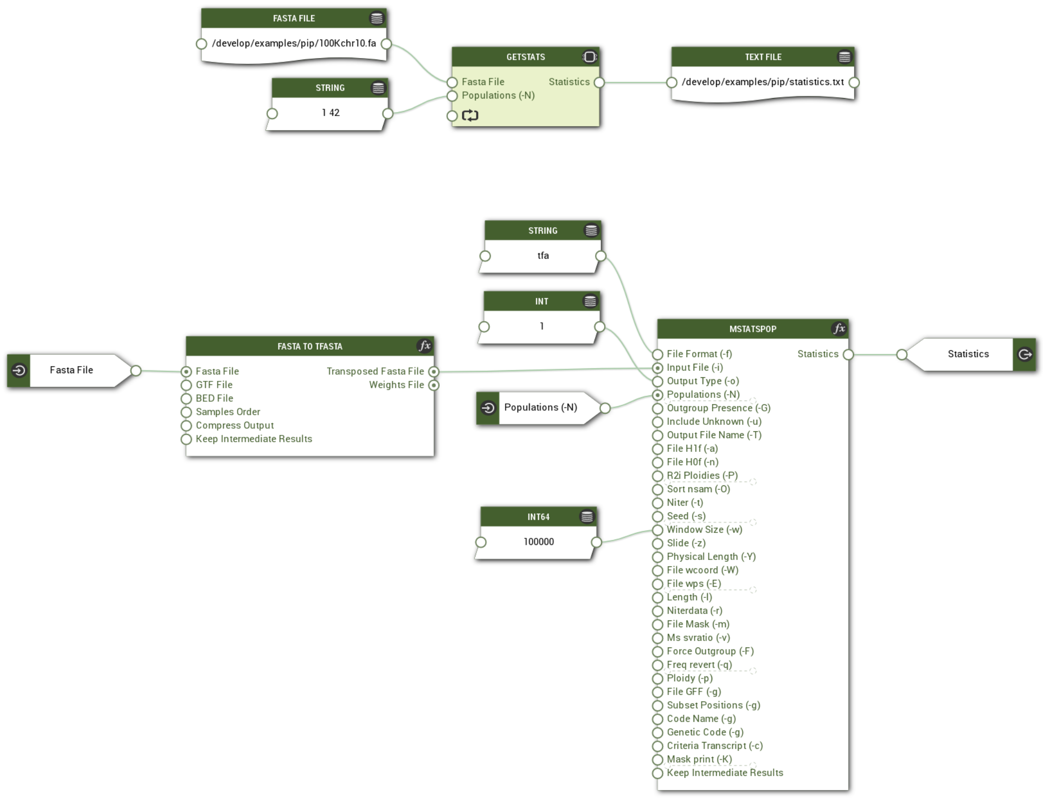


Figure 20.2: ngasp - Experiment / Pipeline Example 2

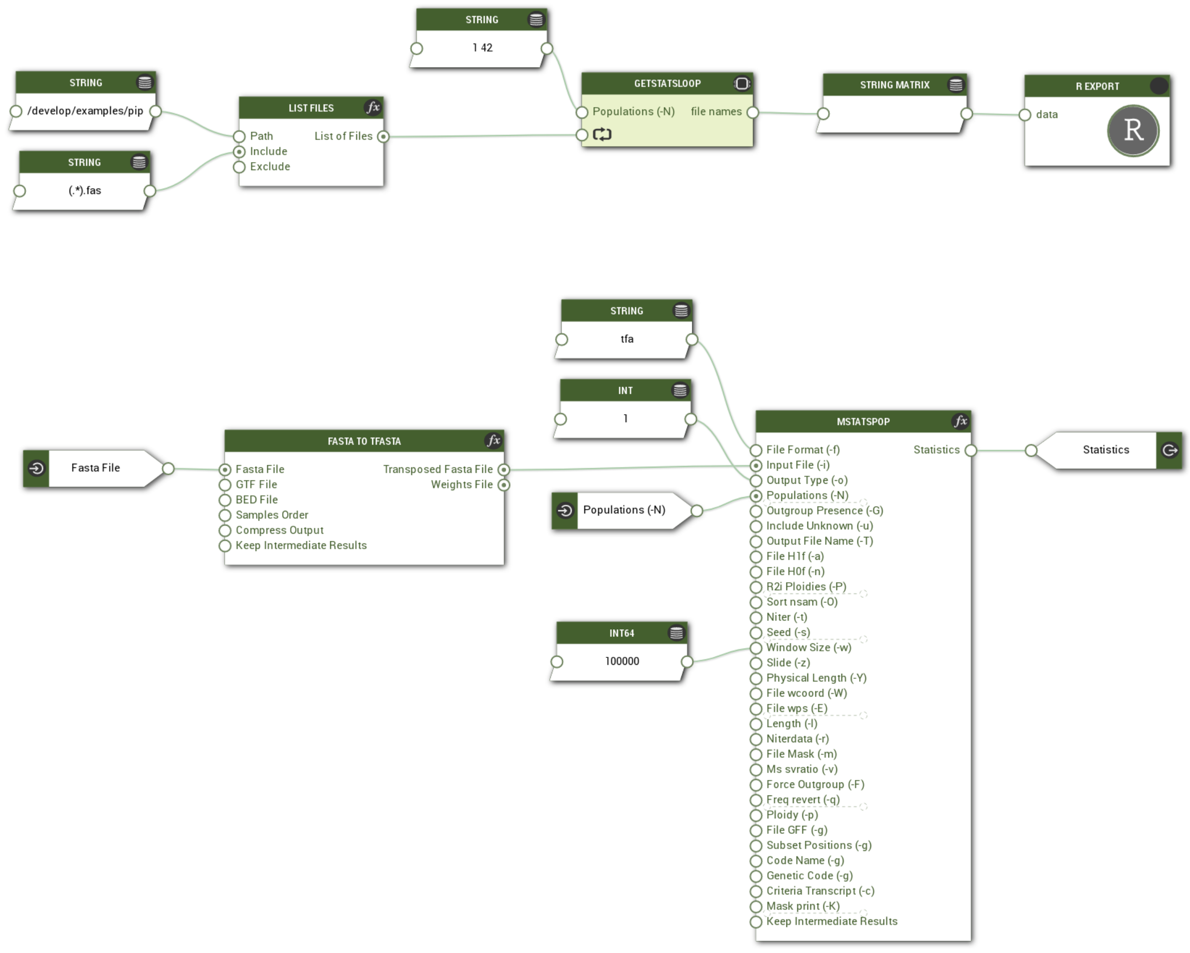


Figure 20.3: ngasp - Experiment / Pipeline Example 3

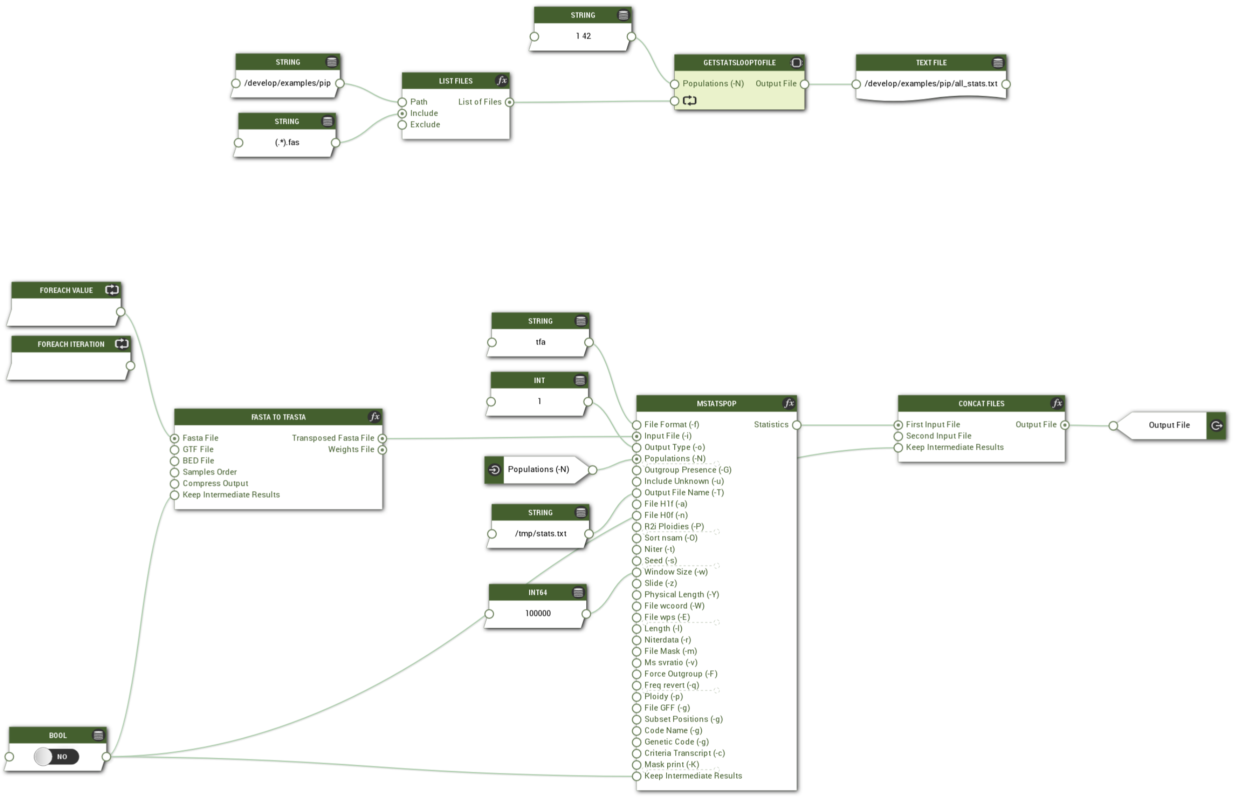


Figure 20.4: ngasp - Experiment / Pipeline Example 4

Part VI

Public Web Site

Chapter 21

Update the website

After reading this chapter you will be able to perform maintenance tasks to the public website.

21.1 Change something

The web site is here: `https://bioinformatics.cragenomica.es/projects/ngaSP`

The source code of this web site is here: `/projects/035-ngaSP/public_web`

You should develop locally: `./ngasp/public-web` and then, upload changes to the remote web site.

21.2 Measure Web Visibility

The web site is analyzed by Google Analytics at `https://analytics.google.com`.

Members:

- sebastian.ramos.onsins@gmail.com
- gonzalo.vera.rodriguez@gmail.com
- joan.jene@cragenomica.es

Glossary of Terms and Acronyms

B

back-end It is the part of the system written in C++ that contains all commands and calculations. It can be executed from a command line and sockets. 7, 9, 10, 12, 13, 38, 40

BAM The BAM format is a binary format for storing sequence data. 26

BED The BED format consists of one line per feature, each containing 3-12 columns of data, plus optional track definition lines. 26

C

calculation It is a C++ function with inputs and outputs.. 12, 13, 24, 26, 31, 32, 38, 45–47

Calculation Developer It is the software developer that uses the system to create new calculations. 12, 35

command It is an ngasp action. . 13, 22, 24, 25

D

Data Manager It is the part of the system that enables the user to share data between calculations. 24

DM Data Manager. 25

Docker It is an open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud. 7, 9, 20–22, 39, 40

Docker Compose It is a tool for defining and running multi-container Docker applications. 9

E

End User It is the user who analyzes his/her data using already available pipelines. 5, 20, 35

experiment It is a set of pipelines (and calculations, also) working together with some input data for a accomplishing a desired purpose. 8

F

Fasta It is a text-based format for representing either nucleotide sequences or peptide sequences, in which nucleotides or amino acids are represented using single-letter codes. 26

front-end It is the part of the system developed in HTML5, JavaScript and NodeJS that implements a graphical interface. 7, 10, 16

G

Git It is a version control system (VCS) for tracking changes in computer files and coordinating work on those files among multiple people. 8, 20, 43

Gradle It is an open source build automation system. 12, 40

gVCF The gVCF is a text file format, stored as a gzip compressed file. It stands for Genomic VCF. A GVCF is a kind of VCF, so the basic format specification is the same as for a regular VCF, but a Genomic VCF contains extra information. 26

J

Jenkins It is a continuous integration software written in Java. 8, 10, 39, 40

job It is a set of tasks. 38–40

M

Mpileup It describes the base-pair information at each chromosomal position. This format facilitates SNP/indel calling and brief alignment viewing by eyes. 26

N

NetBeans It is a software development platform written in Java. 12

ngasp Next generation analysis of sequence polymorphisms. 5, 7, 12, 13, 16, 20–22, 24–27, 29, 30, 45, 51

ngasp script It is a source code file with ngasp's commands and calculations calls. 12

P

pipeline It is a set of calculations working together for a accomplishing a generic task. 8

Pipeline Developer It is the user who develops new analysis pipelines combining the existing available calculations. 35

S

samtools It provides various utilities for manipulating alignments in the SAM format, including sorting, merging, indexing and generating alignments in a per-position format. 7, 13

seqan It is an open source C++ library of efficient algorithms and data structures for the analysis of sequences with the focus on biological data. 8, 13

Software Engineer It is the developer that creates the system and the continuous integration environment. 35

T

TFasta It is a transposed Fasta file. 26

W

Web Developer It is the developer that creates and maintains the project website. 35

X

XCode Xcode is an integrated development environment containing a suite of software development tools developed by Apple for developing software for macOS, iOS, WatchOS and tvOS. 8

XQuartz The XQuartz project is an open-source effort to develop a version of the X.Org X Window System that runs on OS X. 21

Bibliography

[Vicent Driessen's Branching Model] Vincent Driessen, Branching model. `http://nvie.com/posts/a-successful-git-branching-model/` [Online; accessed 16-December-2016].

[GNU Lesser General Public License, version 2.1] GNU Operating System, GNU Lesser General Public License, version 2.1. `https://www.gnu.org/licenses/old-licenses/lgpl-2.1.en.html` [Online; accessed 19-December-2016].

[Pull Request] Aprende GIT, Aprende GIT. `http://aprendegit.com/que-es-un-pull-request` [Online; accessed 8-February-2017].