PIMS:
Pipeline Interface and Management System

User Guide

Joseph Gardner
Department of Genetics
University of Cambridge
UK

Contents

1 Introduction 3

2 User Guide 3

2.1 Running PIMS 3
2.2 Initial window 4
2.3 Adding a new tool 4
2.4 Editing an existing tool 6
2.5 Generating a script 8
2.6 Viewing your existing scripts 10
2.7 Running a script 10

3 Example: Read Trimming and Alignment 11

3.1 Adding the tools 11
3.2 Running the scripts 11
3.3 Assessing the results 11

1 Introduction

PIMS was created to make it easier for those carrying out bioinformatics analysis to create pipelines, to edit them and to record their work. The huge range of bioinformatics tools available to researchers, many of them carrying out the same processes (e.g., sequence alignment) makes it difficult to choose the best tool for your work. In addition, most tools come with a range of adjustable parameters: very useful for tweaking a tool to a particular project, but working out the best parameters *a priori* is difficult (if not impossible). An additional layer of complexity is added when one considers combinations of tools: pipelines that chain together various analyses may give different results with different combinations of tools (each with their own parameter set). Ideally, one would like to systematically test a series of cases and then choose the optimal one, based on some criteria that reflect your project aims. This is doable, but there are challenges. Many of the tools available rely on a command line interface (CLI), which is certainly powerful, but is not the easiest to navigate, especially when one wants to repeatedly change, say, one parameter embedded in the middle of a lengthy command. On top of this, it is crucial to keep an accurate record of the exact settings in any given pipeline run, plus the output at every stage. The sum of the above challenges makes for a daunting task.

PIMS is designed to make this task more manageable. This is achieved in several ways:

- The use of a graphical user interface (GUI) that sets out each tool and its options in a clear way that can be easily understood and edited
- A system for saving bioinformatics tools and their settings for repeated access and reuse, thus saving time when running similar pipelines multiple times
- Automatic record-keeping, including the exact script run, the date and time, and the outputs from every stage of the pipeline

2 User Guide

This guide will go through each part of PIMS and explain its usage. An example can be found in Section 3 of this document.

2.1 Running PIMS

Currently, PIMS is run through the command line:

```
$ python /path/to/pims-vX-X.py
```

Future versions will have something more sophisticated.

One of the first things that PIMS will do is to check if the file system that it uses is present already. This will take the form of a directory in your home folder called “pipeline”, with four subdirectories called “tools”, “config”, “scripts”, and “output”. If any of these are not present they will be created. Currently there is no option for the user to change any of the above (coming in a future version), so if you already have a directory called ”pipeline” in your home folder, it is suggested that you change the name before running PIMS (or edit the source code).

2.2 Initial window

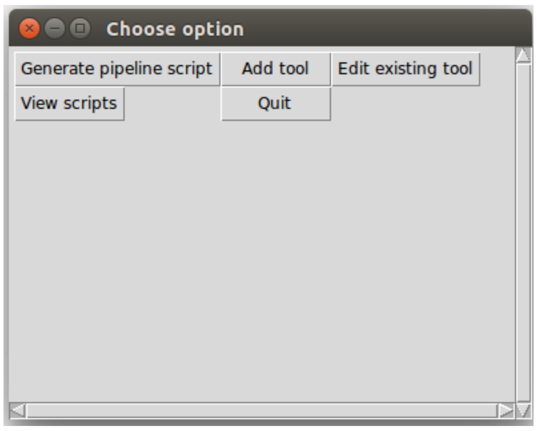


Figure 1: The first window that appears when the program is started.

When PIMS is started, the window shown in Figure 1 will appear. This window will persist throughout the usage of PIMS, primarily because it is used to quit the program, via the “Quit” button. Simply closing the windows will not quit the program properly, and in that case it will be necessary to either use the Unix `kill` command, or to close the Terminal window from which PIMS was started. Aside from “Quit”, the buttons shown are explained in detail below.

2.3 Adding a new tool

Clicking the “Add tool” button in Figure 1 will open the window shown in Figure 2. This window is used to add a bioinformatics tool to the pipeline,

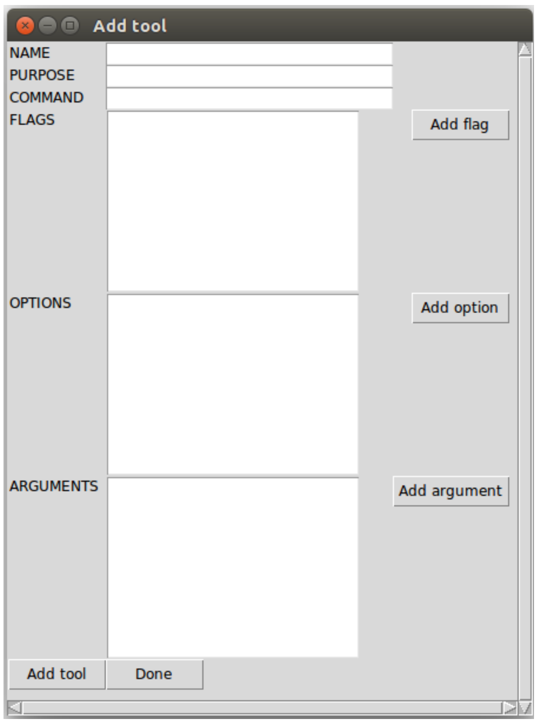


Figure 2: The first window that appears when the program is started.

with the flags, options and arguments of interest, and to then save it to the PIMS file system. The window includes several text input fields, as follows:

- NAME: The name of the bioinformatics tool to be added (e.g., "Bowtie"). This name should be unique, and an error will be raised if you try to add a tool with a name that has already been taken.
- PURPOSE: This field is used to classify the tool being added based on what it does. For example, if the tool was Bowtie, one might put "Aligner" in this field. If you are comparing several tools that do the same thing (e.g., comparing aligners), make sure all the tools to be compared have the same purpose.
- COMMAND: This is the "root" command used to run the tool on the command line, e.g., `samtools view`. This is the command before any options or arguments are added.
- FLAGS: The term "flags" is used to describe optional arguments passed to the tool from the command line that have a name but do *not* take an argument. For example, if we wish to view a SAM file in the Terminal, we might type

```
$ samtools view -S aligned_reads.sam
```

In this case, `samtools view` is the COMMAND, discussed above, and `-S` is a flag. Both single and double hyphen flags are allowed, and the name may contain alphanumeric characters, underscores and hyphens. Illegal characters will raise an error.

- OPTIONS: These are keyword arguments, i.e., a name followed by a value. The following show the allowed formats:
  - `--name=<>`
  - `-name=<>`
  - `--name <>`
  - `-name <>`

  Options will be written to the script exactly as they are typed in this field, with `<>` replaced by the value given for that run, so ensure that the format you put is the correct one for the piece of software.

- ARGUMENTS: These are non-keyword, order-specific arguments that are written to the script after all the OPTIONS and FLAGS. The name given here is not written to the script, and is only to remind the user of the tool's arguments.

The "Add ..." buttons on the right hand on the window can be used to add flags, options, and arguments to that tool's list in groups. Those entered into the box already will be saved and the box will be emptied, so that users entering long lists can keep track of what has been added.

Once the relevant fields have been completed, pressing "Add tool" will save the tool in the filesystem for later use. The fields will then be emptied, ready for the next tool. Note that "Done" will *not* save the current contents of the fields, but will simply close the window.

2.4 Editing an existing tool

Once a tool has been added to the system, it can be edited using the "Edit existing tool" button in Figure 1. This opens the window shown in Figure 3. The drop-down menu will contain all tools that the user has added so far. The user can alternatively type in the name of the tool, and an error will be raised if it is not in the list. Clicking "Go" will open the window shown in Figure 4. In this window, the information previously input about the selected

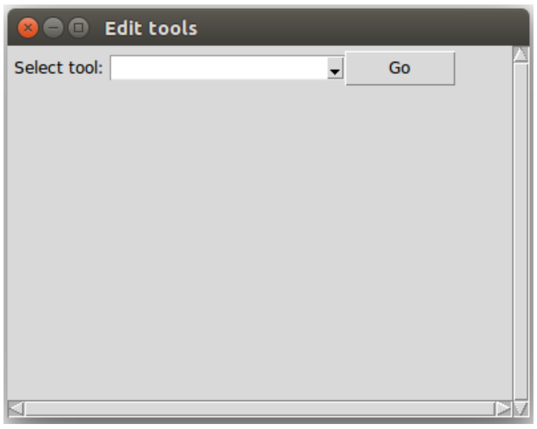
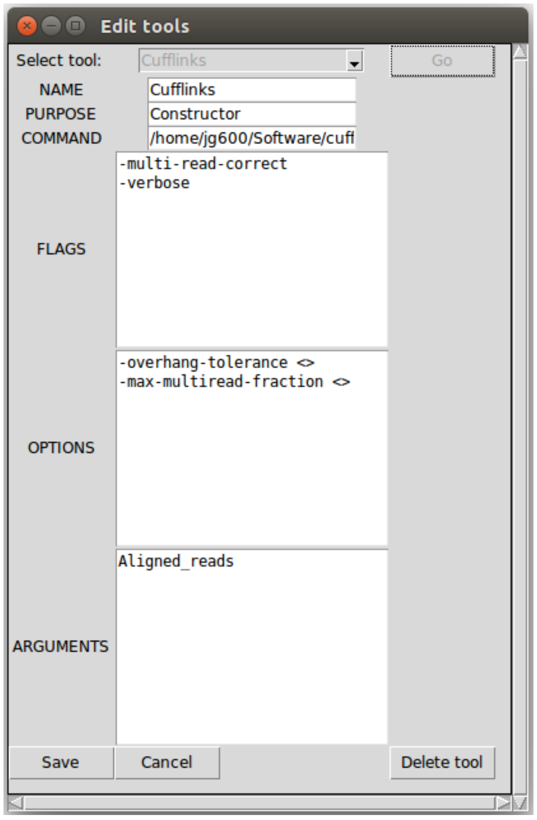


Figure 3: Here, the user chooses which tool to edit from those that have been added.



NAME	Cufflinks
PURPOSE	Constructor
COMMAND	/home/jg600/Software/cuff

```
-multi-read-correct
-verbose
```

```
-overhang-tolerance <>
-max-multiread-fraction <>
```

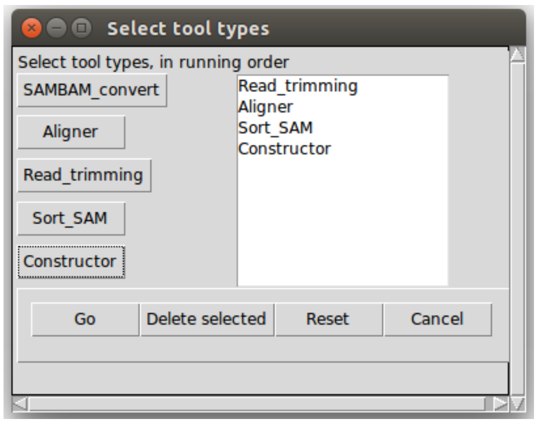
```
Aligned_reads
```

Figure 4: Here, the user can change the previously entered information about a given tool.

tool will be displayed, and can be edited and then saved. The tool can also be deleted permanently.

2.5 Generating a script

If at least one tool has been entered and saved, a script can be generated. Clicking the "Generate pipeline script" button in Figure 1 will open the window in Figure 5. Types of tool should be selected in order of execution



Read_trimming
Aligner
Sort_SAM
Constructor

Figure 5: The types of tool are selected in the order in which they should be executed.

by clicking the buttons on the left-hand side of this window. Once all the desired types have been selected (not necessarily all of them), clicking "Go" will open the window in Figure 6. Here, the user can select the flags to be included and give values to the options and arguments of each tool selected in Figure 5.

As shown in Figure 6, each tool has a frame that contains the various checkbox and entry fields. Clicking the background of a tool's frame will "activate" it, so that it is included in the final script. Either 0 or 1 tools can be active per purpose. Fields can only be filled while the tool is active, and the contents will not be lost if the tool is made inactive.

In many cases, the same or similar sets of values will be used in several runs. To accommodate this, configurations can be saved and loaded repeatedly using the 'Save configuration" and "Load configuration" buttons at the bottom of the window in Figure 6.

Once the various FLAGS, OPTIONS, and ARGUMENTS have been given, the user can generate a script. Clicking the "Run" button in the bottom left corner of Figure 6 will open the window shown in Figure 7. In this window, the user names the script and can add optional notes, which

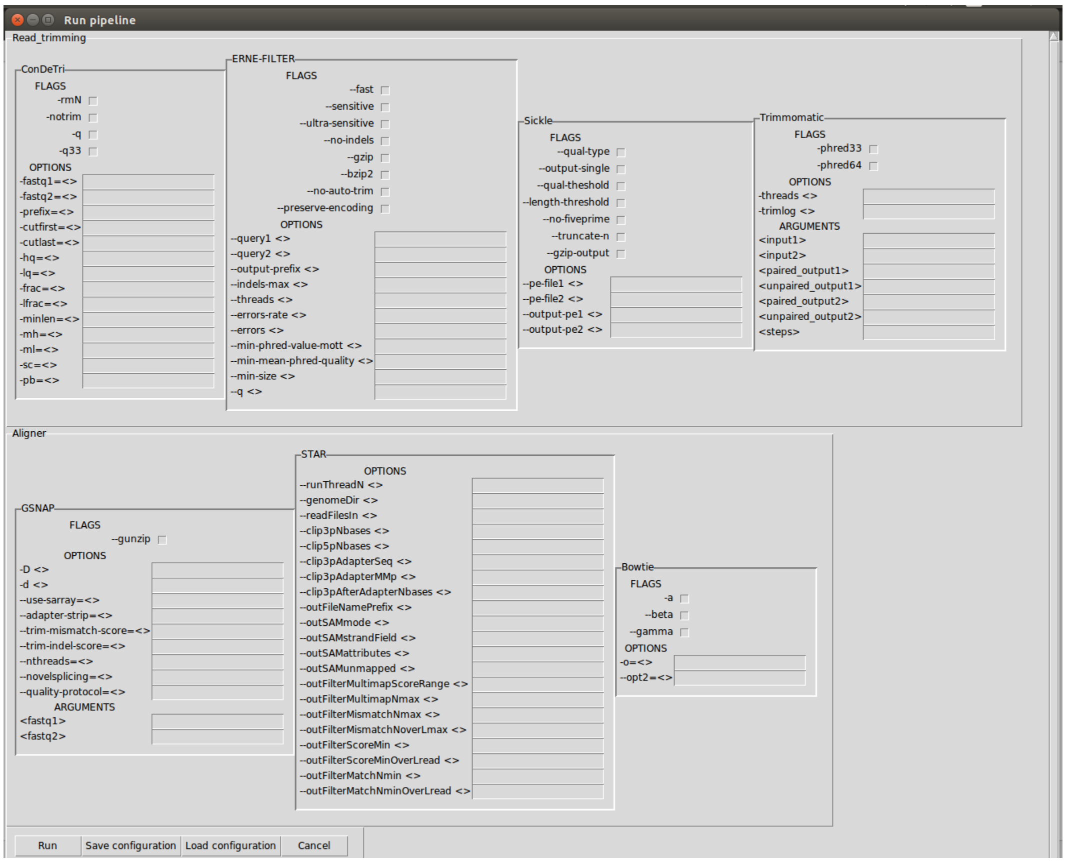


Figure 6: The window used to give values and arguments for a given pipeline run.

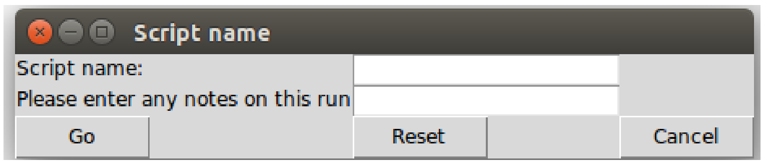


Figure 7: The window used to name a script and attach any notes about it.

will be saved in the directory created when the script is run. The script will carry out the following steps:

1. Create a directory named with the script name and the date and time of the script's creation (in the next version this will change to the time of running)

2. Make a copy of itself in the new directory, also labelled with the time of creation

3. Write the user's notes on the script to a file called "NOTE" in the new directory
4. Run the tools that were selected for this run from inside the new directory
5. Delete itself (so the only copy is within the newly created directory)

The end result will be a directory containing the script, the user's notes, and all of the intermediate and output files from the various tools[1].

2.6 Viewing your existing scripts

Scripts created using PIMS will be saved in `pipeline/scripts`. These can be viewed by clicking on the "View scripts" button in Figure 1, which will open the window in Figure 8. The drop-down menu is used to select an

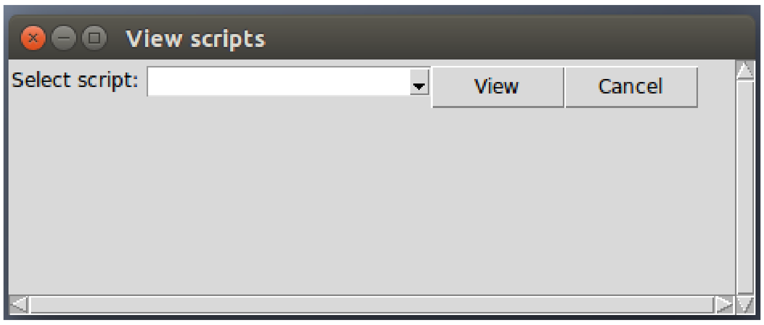


Figure 8: The window used to view scripts that have already been created.

existing script, and clicking "View" will display it. This cannot be used to edit scripts, and is probably not the best way of viewing them. This functionality is only intended for users to be able to remind themselves of what they have already done, and any more complex tasks are best left to the user's favourite editor.

2.7 Running a script

The scripts generated are compatible with Unix systems only (future versions may include Windows compatibility). Users should check their script before running to ensure that it will do what they expect. Scripts can be run from the command line in the following fashion:

[1]Unless the user has changed the tool parameters so that the output directory is not the current working directory.

```
$ bash name.script
```

There is not currently any sophisticated error-handling built into the scripts, so users should make sure that all paths, commands, flags, etc. are correctly typed before running. If the pipeline includes several steps that take a long time, it is recommended that the process is checked frequently to make sure it has not exited with an error.

3 Example: Read Trimming and Alignment

Read trimming is a widely used pre-analysis step in many bioinformatics pipelines. In high throughput sequencing experiments, the sample to be sequenced is broken up into short fragments which are then sequenced. The output of the sequencing machine is a set of short sequences that must be processed in order to produce a full picture of the sequence in the sample. This processing is usually either alignment to a reference genome (or transcriptome), or *de novo* assembly.

Unfortunately, sequencing machines are not perfect, and will occasionally mislabel a base. Incorrect bases will lead to incorrect alignments and assemblies, impacting the quality of the results. To allow researchers to deal with this, sequencing machines usually assign each base a quality score (or *phred score*) that reflects the probability of the base call being correct. A higher score means a lower probability of a mistake. Trimming and filtering algorithms use these scores, and either trim low quality bases off the ends of reads, or remove reads entirely if they are deemed to be of too low quality throughout.

While trimming is undoubtedly important when dealing with sequencing data, there is debate about how much trimming is actually useful. There are also many available trimming algorithms, each using different methods and demonstrating different results, as well as different impacts on the downstream analysis steps (e.g., alignment). I will therefore use read trimming and alignment as a demonstration of the use of PIMS.

3.1 Adding the tools

3.2 Running the scripts

3.3 Assessing the results