A peer-reviewed version of this preprint was published in PeerJ on 21 November 2019.

View the peer-reviewed version (peerj.com/articles/8111), which is the preferred citable publication unless you specifically need to cite this preprint.

Joppich M, Zimmer R. 2019. From command-line bioinformatics to bioGUI. PeerJ 7:e8111 https://doi.org/10.7717/peerj.8111

From command-line bioinformatics to bioGUI

Markus Joppich[1] **and Ralf Zimmer**[1]

[1]**Institute for Informatics, LFE Bioinformatik, LMU München, München, Germany**

Corresponding author:
Markus Joppich[1]

Email address: joppich@bio.ifi.lmu.de

ABSTRACT

Bioinformatics is a highly interdisciplinary field providing applications for scientists from many disciplines. Installing and starting applications on the command-line (CL) is a problem for many scientists, nonetheless, most methods are implemented with a command-line interface only. Providing a graphical user-interface (GUI) for bioinformatics applications is one step towards routinely making CL-only applications available to more scientists, and, thus towards a more effective interdisciplinary work.
We identified two main problems for conveniently using CL bioinformatic tools: First, many tools work on UNIX-systems only, while many scientists use Microsoft Windows. Second, scientists refrain from using command-line tools which, however, could well support them in their research. Both issues are addressed by bioGUI.
The *bioGUI* framework provides easy means to make CL tools available for most scientists, especially making use of Windows Sub-system for Linux (WSL), which provides a *native* Ubuntu bash on Windows. In addition, *bioGUI* templates are easily created (automatically), making this framework highly rewarding for developers. The *bioGUI* repository allows to install and use bioinformatics tools with just a few clicks.

INTRODUCTION

Many advances in bioinformatics rely on sophisticated application software. Examples are Trinity (Grabherr et al. (2011)) for *de novo* assembly in conjunction with Trimmomatic (Bolger et al. (2014)), or the HISAT2, StringTie and Ballgown pipeline for transcript-level expression analysis (Pertea et al., 2016). These tools have in common, that, locally installed, only a command-line interface (CLI) is provided, implying a burden for many non-computer affine users. Jellyfish(Marçais and Kingsford, 2011), Glimmer (Delcher et al., 2007) and HMMer`http://hmmer.org` natively run only in UNIX-environments and require a sophisticated setup on Windows. In addition, the installation of command-line (CL) tools is a challenge for non-computer specialists e.g. due to package dependency resolution. This problem has recently been addressed by the AlgoRun package (Hosny et al. (2016)), providing a *docker*-based repository of tools. Being a web-based service, it is limited to web-applicable data sizes or local data must be made available to the docker container. While AlgoRun has the advantage of processing data anywhere, it relies on *docker* which needs a more sophisticated setup on Windows than downloading an app from the Microsoft Store, like Windows Sub-system for Linux (WSL), in order to run with local data. More importantly, a cloud-based service may conflict with data privacy guide lines (Schadt (2012)), e.g. with respect to de-anonymisation of patient samples (Gymrek et al. (2013)).

A frequent argument for not providing a graphical user-interface (GUI) is the overhead for developing it and the effort to make it really "user friendly". Often GUIs are deemed unnecessary by application developers. However, one can be sceptical whether non-computer-affine scientists can efficiently use CLIs in their research. In fact, Albert (2016) notes that "Bioinformatics, unfortunately, has quite the number of methods that represent the disconnect of the Ivory Tower". Pavelin et al. (2012) note that software is often developed without any focus on usability of interfaces (for end-users). While this does not imply that any GUI is helpful, we argue that without a GUI, there is almost no chance that CL programs are of best help for scientists. Besides, a GUI is often more convenient and helps to avoid parameter errors, especially when not yet running a software routinely on many data. Smith (2013) also states that

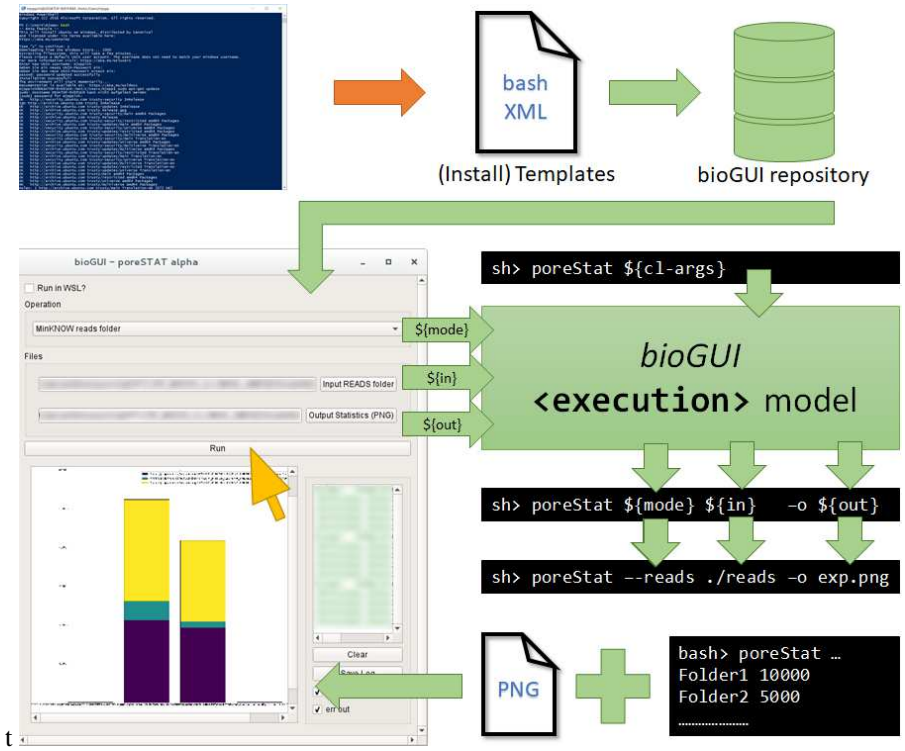


Figure 1. Only little human interaction is needed to run a CL application with a *bioGUI* template. An (install) template has to be submitted to the *bioGUI* repository by a developer (orange) once. From there, users (yellow) can download desired templates and install applications from the *bioGUI* (green) repository. The template is then rendered as input mask in *bioGUI*. Upon selecting the input for the application using the GUI, *bioGUI* constructs the command-line arguments to execute from the input and presents output to the user. Simple output, text or images, is directly displayed in *bioGUI*.

GUI-driven applications make daily work in biology or medical labs easier. Smith remarks that many end-users have a “penchant for point and click”. Therefore these may not be able to effectively use CL tools. Still they should have the ability to access and analyse their own data. Many proprietary software solutions address this demand: they allow GUI-based data management, while also being extensible via plug-ins. Smith (2015) points out that one of the biggest advantage of such plugins is to combine the power of peer-reviewed algorithms with a user-friendly GUI. Thus, providing a GUI is an important step towards the applicability of methods by end-users. Visne et al. (2009) present a universal GUI for R aiming to close the gap between R developers and GUI-dependent users with limited R scripting skills. Additionally, web-based workflow systems, like Galaxy (Afgan et al., 2016) or Yabi (Hunter et al., 2012) provide means to easily execute programs, but aim at more complex workflows. However, these tools are designed to be run and maintained by bioinformaticians for several users and are not meant to run on an single, individual basis, like in small labs.

In recent Windows operating systems the WSL feature can be activated. This feature provides a native, non-virtualised Ubuntu environment on Windows, allowing to run most applications that also run on Ubuntu. The remaining problem for scientists is the installation and usage of CL applications.

Here, *bioGUI*, an open-source cross-platform framework for easily running CL applications from a GUI, is presented. It uses a XML-based domain-specific language (DSL) for template definition, which lowers the initial effort to create a GUI. Creating a *bioGUI* template for a CL application usually takes only minutes. Install modules provide an easy method to download and install bioinformatic applications on WSL on Windows, Mac OS and Linux. *bioGUI* also addresses data management by saving filled out templates, enabling easy reproducibility of data analysis (Figure 1).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018

METHODS

For developing *bioGUI* we employed a use-case driven feature development phase before further going into the development of *bioGUI*. This section first summarizes existing systems, then covers the use-case study we performed and goes into detail of how *bioGUI* works.

Existing workflow systems

There are several workflow systems already available. Most prominent in bioinformatics are the Galaxy server and Yabi. In addition, workflow specification languages such as Common Workflow Language (CWL) exist.

Galaxy & Yabi The Galaxy server is a well known workflow system in bioinformatics (Afgan et al., 2016). While bioGUI does not aim to be a workflow system as Galaxy with, for instance, data management, there are similarities. For instance, Galaxy also provides a GUI for the added workflows. However, all data to be processed by Galaxy must either be on the server itself or uploaded to a location that is reachable by the server. Galaxy can access cloud storages, but highly sensible data may not be uploaded to such storages as pointed out in the introduction. Additionally, Galaxy requires UNIX knowledge to be installed and does not provide a binary for installation. Galaxy is not cross-platform compatible (Windows is supported through WSL but still requires UNIX knowledge). However, users provide docker container for Galaxy, where a local storage can be mounted.

Another framework providing similar options is Yabi (Hunter et al., 2012). Yabi however is only distributed using Docker container.

Common Workflow Language (CWL) The Common Workflow Language (CWL) (Amstutz et al., 2016) is a new standard to define workflows. In this standard, inputs, input-types as well as the corresponding parameters are stored. Additionally, inputs can have a help text included.

Using the bio.tools ToolDog software (Hillion et al., 2017), CWL workflows can be generated and exported for many bioinformatics software. An advantage of using bio.tools is the automatic annotation and description of input and outputs. Unfortunately for many packages no CWL workflows have been deposited.

Use-case study

One of the main goals for developing bioGUI is to create a powerful framework, which is easy-to-use for scientists/users and which does not create significant overhead for the application developer.

In order to illustrate this, we introduce two classes of possible users: The first class represents a typical wetlab scientist (e.g. not a computer scientist) who is performing a research task, e.g. a sequencing project. This class will later be referred to as *the user*. The second class pictures a software developer releasing an application using a new algorithm to solve the alignment of sequencing reads. This class thus depicts a typical *developer*.

From the two use-cases (see also Appendix A.1) we identify several requirements/goals for *bioGUI*:

1. installing new programs must be simple and should not require system administrators
2. creating a GUI for a program must not take a lot of time
3. templates must bring a basic GUI to run the programs, output must not be interpreted
4. templates must be saveable for later re-use and reference, and also searchable
5. the system must be lightweight (runtime overhead, disk-space) to even run on laptops
6. installing a program may require additional (protected) external files

Finally we developed a paper mockup with which we went through the anticipated workflow of the user. We identified several input components and features the *bioGUI* program has to include (Figure 2).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018

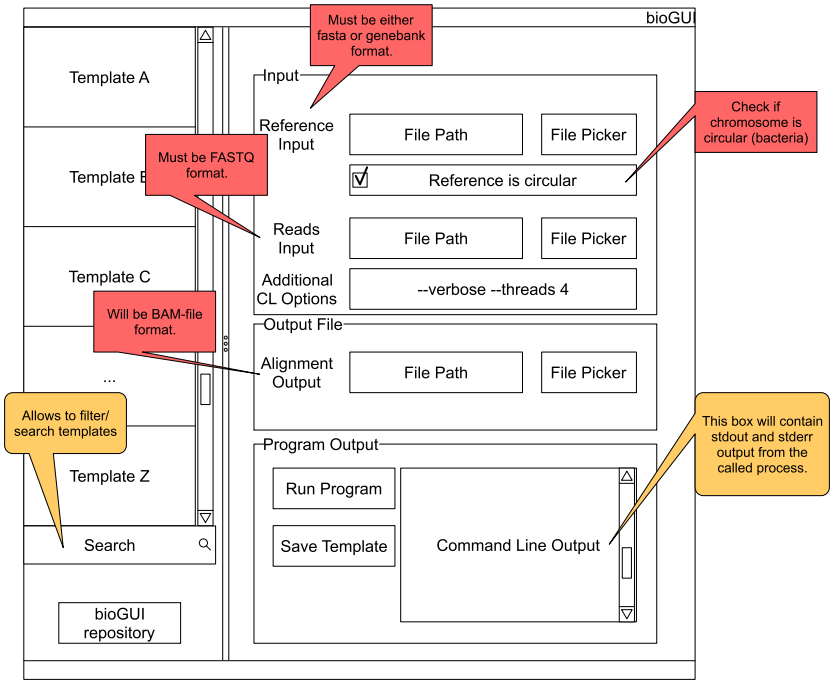


Figure 2. *bioGUI* mockup showing the elements of which a template could be made of. The GUI has a searchable list of installed templates as well as a link to our repository of templates. The right side is reserved to the currently displayed GUI. Here the user can have a structured few of the required options to be set, as well as tips on how to fill these options (tooltips in red). Finally, the user has the possibility to run the program by clicking a button and seeing the program's output.

bioGUI approach

During the use-case study we found two main problems with bioinformatics software for scientists: first the installation of potentially useful software and second its usage. Both problems have in common, that they are expected to be performed on the command-line line. A GUI for achieving the respective task is missing.

Especially the first problem, installing bioinformatics software on a user's machine, poses a few problems. Most bioinformatics software is written for a UNIX operating system, while in general Microsoft Windows is the dominant operating system. In order to overcome this problem, *bioGUI* makes use of Windows Sub-system for Linux on Windows. Besides, there are scientists using Mac OS or Linux systems. In order to support all these users, *bioGUI* uses a cross-platform approach. *bioGUI* is developed in C++ using the Qt framework.

The general workflow for any program using *bioGUI* is shown in Figure 1. For a CL application, the specific template must be written by the software developer (orange) in a XML-based DSL and can then be made available in the *bioGUI repository* (green). Such templates can be automatically retrieved by *bioGUI*. Upon selection by the user, *bioGUI* displays the input mask as defined in the template. When the user (yellow) has filled in all parameters, the parameters are collected by *bioGUI* and assembled into command-line arguments which are used to execute the original CL-only application. Upon completion, simple results (like text-output or images) can be shown in *bioGUI* directly, or an external application is opened.

This section will first describe how *install modules* are implemented, how *bioGUI templates* work and finally, how these are made available to a broad public.

Install Modules

Install modules (iMods) are designed to install software such that *bioGUI* can access the software. Essentially, iMods are bash scripts which allow an automatic installation of software into a predefined

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018

location. For this purpose, iMods receive several arguments from *bioGUI* when launched, e.g. the install location where to place the software, the *sudo* password to fetch packages via a system's package manager (e.g. aptitude, conda, ...), whether the software should be made available to the user via the system's PATH variable.

iMods download and install software and make it available to the user and bioGUI. However, some software cannot be simply downloaded, but is distributed by installers. For this purpose, the install module template can be extended by further input fields. These must be specified by bioGUI elements and their values are added to the end of the command-line arguments of the install module. An iMod can then execute the referenced installer.

Finally, an install module should contain the specification of its *bioGUI* template and could hard-code the path to the installed the software as well as other default values that can already be derived during the installation, before sending it back to *bioGUI*, which stores the template in its database.

bioGUI Templates

bioGUI templates are the actual end-user-interface to programs. The *bioGUI* templates define the look & feel of the user-interface for program execution and also define how CLI-programs are called (with corresponding parameters).

Each *bioGUI* template consists of two parts (Figure 3). The first part (`<window>` model) defines the visual appearance of the GUI. The second part (`<execution>` model) defines the processing logic of the template. Input values from the GUI components are collected and assembled (e.g. pre-/post-processing steps) to call command-line applications. As part of this assembly, input values from the GUI may be transformed using (multiple) predefined nodes. Concatenations are possible using the `<add>` node, and constant values can be inserted using the `<const>` node. System environment properties, such as the operating system, the computer's IP address or specific directories can be collected using the `<env>` node.If the regular nodes are not sufficient, e.g. because more complex string manipulations should be made (see use-case study), *script* nodes may also accept functions written in *LUA* or *javascript*.

In general, the execution part infers a network with inputs (e.g. GUI elements, other nodes within the execution part) and actions (if, add, ...). As example, the execution network (which resembles a petri net) for some template is exemplarily shown in Figure 4.

bioGUI Integreation

Unfortunately CWL only describes the command-line workflow and neither provides a GUI nor means to install the desired tool. Due to this more general specification, CWL fits most problems, but specific annotations of inputs, explanations, the embedding of images is not supported in CWL.

While developers can always create templates manually, *bioGUI* supports developers by offering a template generator from CWL templates or python argparse parsers. Since there are already many CWL templates available for bioinformatic CL applications, CWL files can be used as a base to automatically generate *bioGUI* templates from. Using the *bioGUI template generator* for argparse, it is also possible to automatically generate templates from CWL files (making use of the cwl2argparse program provided by CWL). Our generator takes as input the argparse parser or CWL file and creates input elements for all elements. In case the type of an input is unclear or not supported, the generator falls back to a regular text-input element.

RESULTS AND DISCUSSION

Here we present *bioGUI*, a framework for easy GUI provision in the life sciences. The focus is to enable as many researchers as possible to use high-quality command-line programs, by especially lowering the hurdles to overcome for using bioinformatics software on Windows. Finally, we want to discuss the steps taken for developing *bioGUI*.

Use-case analysis

The use-case analysis has revealed several requirements for *bioGUI* to enable the user to perform the sequencing analysis and to allow the developer a fast template creation (Figure 5).

An easy installation (goal 1) is given through the availability of install modules, which can be downloaded from the *bioGUI repository*. These also allow additional inputs (e.g. python wheels for albacore, goal 6).

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018

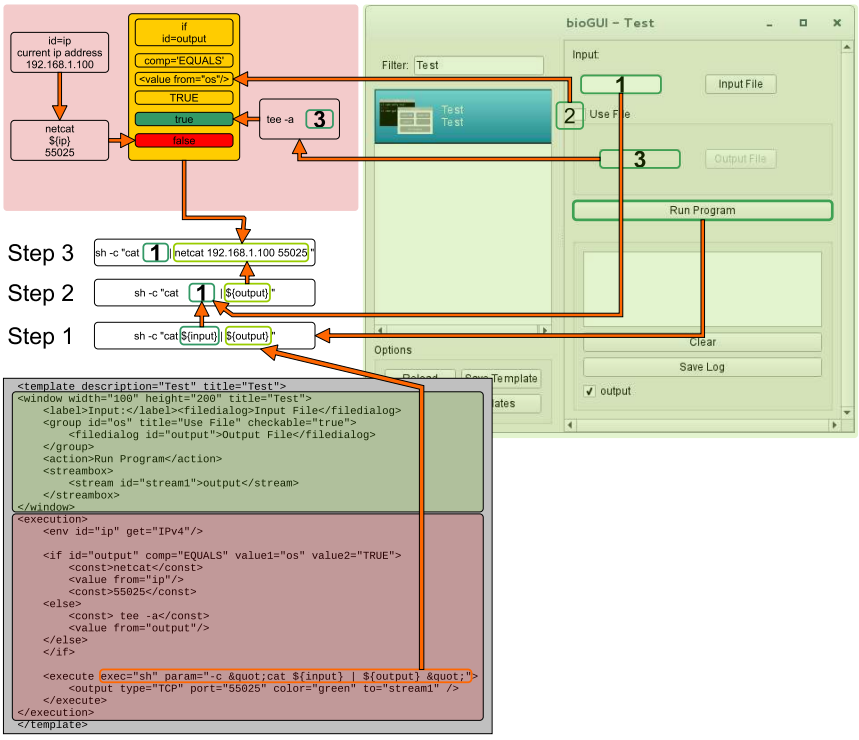


Figure 3. Template construction and evaluation in *bioGUI*. First, the green *window* part is used to create the graphical interface. Once the user clicks the *run* button, the execution part of the template (red) is executed by constructing and executing the appropriate system call. This system call is constructed in 3 steps by replacing variables with evaluated terms from the user's input.

The install modules combine the installation of an application and the creation of the actual GUI template. The former part can be essentially copy & paste from the continuous integration (CI) container setup (goal 2). Even if not, most bioinformaticians extensively use Ubuntu and/or bash-scripts. Thus writing a script to install dependencies is not a significantly hard workload. We have reached a seamless and time-efficient creation of templates using an XML-based domain-specific language. XML is particularly helpful as it allows to specify hierarchies and attributes to objects natively. Using our template generator for CWL and python-argparse, *bioGUI* templates can be created even faster (goal 2). The templates are highly flexible in the creation of command-line parameters also due to providing script nodes.

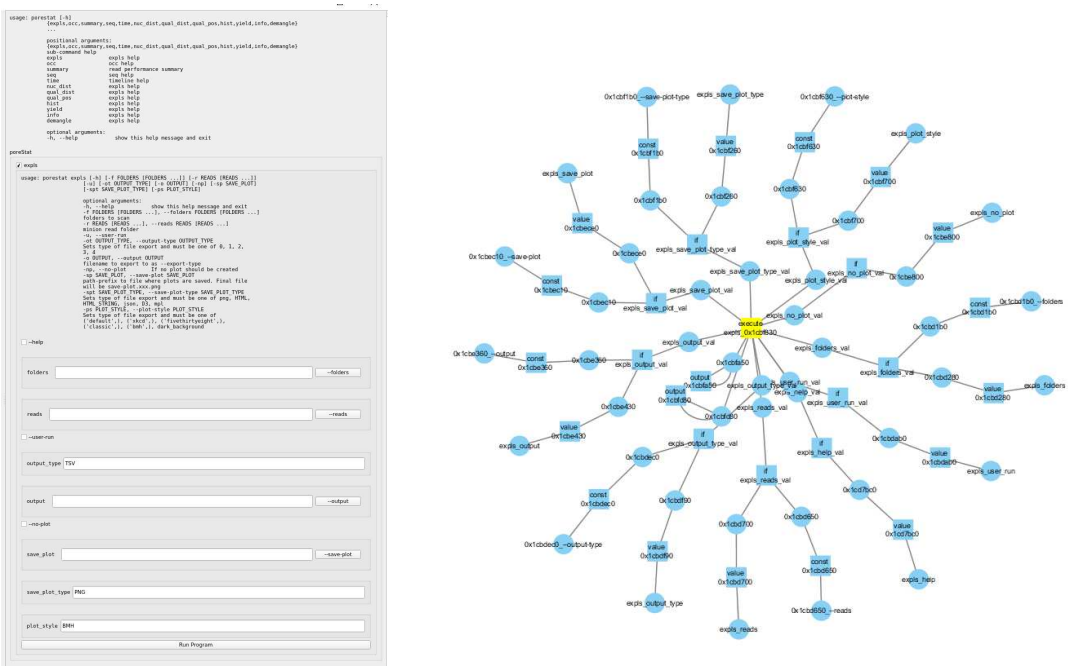
By providing install modules and templates, it becomes feasable for most scientists to install and use high-quality open-source bioinformatic tools.

The *bioGUI* program is cross-platform compatible and only requires a few MB of disk space (goal 5). *bioGUI* implements several possibilities to execute applications. In general, the only runtime overhead involved is the creation of a bash-process which starts the actual program with the calculated command-line arguments (goal 5). *bioGUI*, being a local stand-alone application, has the possibility to use both, locally installed and web-based applications, reachable within a controlled environment and with large data.

The user interface can be made easily understandable (goal 3). Using text-labels, the user can get help (if given by the developer), links can be used to provide further information and most importantly tool-tips could also hint the user to which information is needed at a certain step.

Finally, filled out templates can be saved, and all available templates can be filtered (goal 4). This enables to keep track of performed analyses, and makes results more reproducable, because parameters are saved. Additionally, using the bioGUI repository, templates can easily be shared among users, making it easier to standardize runs among different users or even institutes.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018



(a) Automatically generated template.

(b) Resulting execution network.

Figure 4. An automatically generated *bioGUI* template from the poreSTAT (Internal tool for minION sequencing analysis) python argument parser (a). The resulting execution network for the *bioGIU* template is shown in (b). The central node represents the fully assembled command-line argument (yellow).

bioGUI repository

We provide a repository of pre-configured templates on our website (Figure 6), where authors and users can search for and browse existing templates, or submit new ones. *bioGUI* can access uploaded templates and save them directly for use. Specifically for WSL and Ubuntu users, install modules are provided, which take care for dependency resolution and install applications (locally) into the user's home. This currently works in any environment using *apt* package manager, but users can submit templates which also support other environments, since install modules are bash scripts.

Install modules download and potentially severely alter a system (especially if the *sudo* password is supplied). Thus submitted install modules are manually curated and are only accessible when no security threat has been identified.

One major goal of *bioGUI* is to enable any scientist to use bioinformatics software. While we extend the repository on a regular basis depending on our own use, users can also request new templates for software relevant to them.

Availability & Extensibility

bioGUI is open software and users or institutions can either use our global bioGUI repository or make bioGUI use a specific repository, e.g. one which is only reachable within an institution.

The source code for bioGUI can be accessed on GitHub `http://github.com/mjoppich/bioGUI`. While a number of use cases and corresponding components are already included in bioGUI, we would encourage users to contribute on github by either pushing their own extensions, or opening feature requests. More documentation (installation & setup guide, how to write templates) is also available via GitHub. Supporting tools (cwl2biogui) are available from the tools repository `http://github.com/mjoppich/bioGUI`.

Conclusion

The *bioGUI* framework makes it easy to develop GUIs for CL applications. Having a GUI lowers the anxiety for scientists to use bioinformatic applications. *bioGUI* accomplishes this by its templates and

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018

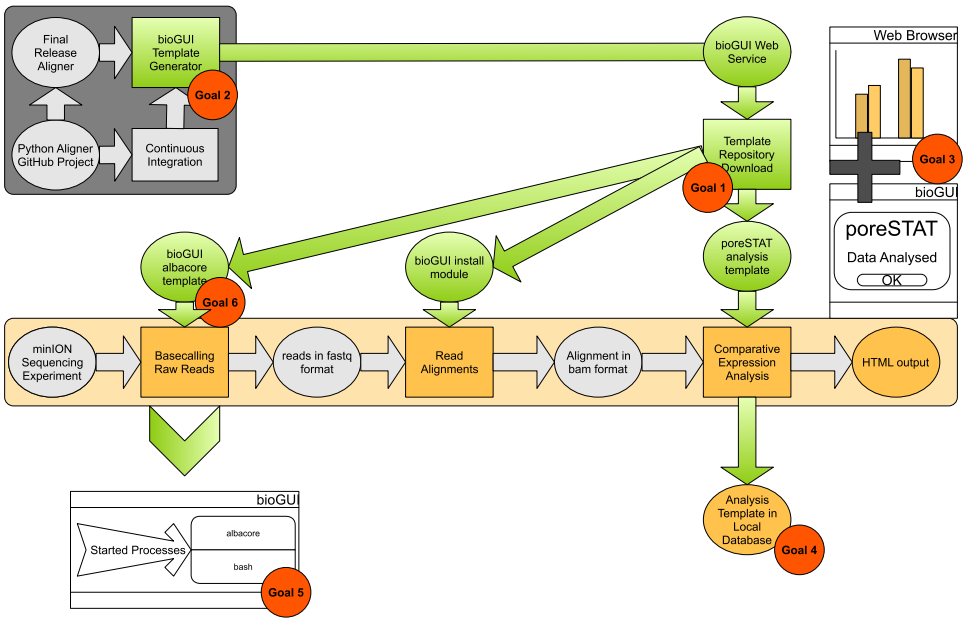


Figure 5. *bioGUI* use-case study workflow analysis. The dark-gray underlayed tasks represent a developer's use-case, and the bright-yellow part represents the analysis pipeline the user wants to execute. All tasks (rectangles) with required user-action are shown in orange. Tasks and results prepared by *bioGUI* are shown in green.

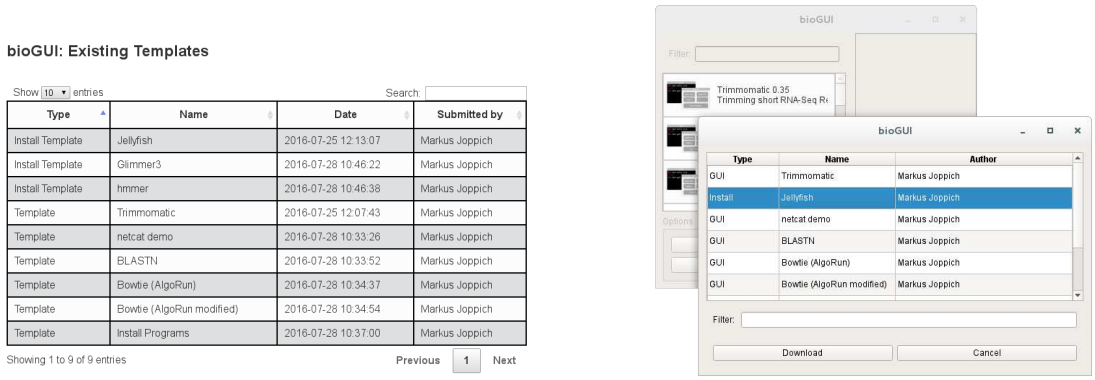
install modules. Thus, an easily creatable, cross-platform GUI experience for many open-source bioinformatics applications is provided through *bioGUI*. In particular, with *bioGUI*, academic bioinformatics applications can easily be deployed to Windows workstations and laptops.

Due to the separation of the GUI components and the program logic, templates can be created in only two steps. First, the template developer adds input elements to the window and, second, assembles these inputs according to the needs of the application back into command-line arguments. This way almost any command-line application can be used with a GUI, enabling many more researchers to use high-quality open-source tools. Providing install modules to make UNIX applications available to Windows users (via WSL) supports this goal.

bioGUI can not replace a dedicated GUI for all cases. A hand-tailored user interface (UI) will still be better usable and more user-friendly than any generic solution can be. This was shown in our use-case re-iterations, where certain programs require special solutions, let alone from displaying or interpreting the results. However, especially with the install module concept we aim to provide a seamless installation and create the possibility to run CL applications by scientists. Our use-case has shown that a GUI indeed lowers the problems scientists have with Unix CL applications and makes execution and usage of these tools possible and/or more comfortable.

Since sequence analysis became a de-facto standard even in wetlabs, users with limited experience in using CL applications and their installation should be able to perform their analysis quickly and reliably. Unfortunately, bioinformatics is a discipline where this potential is rarely used. Using *bioGUI* it becomes a simple exercise to use *trimmomatic* or any other CL tool, while currently running trimmomatic is recognized as "as user unfriendly as it gets" (Albert, 2016). Currently, there are already templates for twelve tools in our *bioGUI* repository. Thus, *bioGUI* lowers the burden to use excellent tools, allowing more scientists a better analysis of their data. With *bioGUI* it must not be first understood how to use and navigate on the CL, instead, the focus is set on the applications, its method and parameters, and finally the data.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018



(a) On our website a list of already existing templates can be browsed. Besides the description and author, also the type (install module or template) is shown.

(b) All uploaded templates can be downloaded directly from within bioGUI. bioGUI allows to search for install modules and templates.

Figure 6. *bioGUI* repository browser on our website (a) and in *bioGUI* (b).

ACKNOWLEDGMENTS

We thank Luisa Jimenez-Soto and Gergely Csaba for their valuable input on how to improve this software and for reviewing the manuscript.

REFERENCES

Afgan, E., Baker, D., van den Beek, M., Blankenberg, D., Bouvier, D., Čech, M., Chilton, J., Clements, D., Coraor, N., Eberhard, C., Grüning, B., Guerler, A., Hillman-Jackson, J., Von Kuster, G., Rasche, E., Soranzo, N., Turaga, N., Taylor, J., Nekrutenko, A., and Goecks, J. (2016). The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic acids research*, 44(W1):W3–W10.

Albert, I. (2016). *The Biostar Handbook*. Albert, Istvan.

Amstutz, P., Andeer, R., Chapman, B., Chilton, J., Crusoe, M. R., Valls Guimer, R., Carrasco Hernandez, G., Ivkovic, S., Kartashov, A., Kern, J., and al., E. (2016). Common Workflow Language, draft 3.

Bolger, A. M., Lohse, M., and Usadel, B. (2014). Trimmomatic: a flexible trimmer for Illumina sequence data. *Bioinformatics*, 30(15):2114–2120.

Delcher, A. L., Bratke, K. A., Powers, E. C., and Salzberg, S. L. (2007). Identifying bacterial genes and endosymbiont DNA with Glimmer. *Bioinformatics (Oxford, England)*, 23(6):673–9.

Grabherr, M. G., Haas, B. J., Yassour, M., Levin, J. Z., Thompson, D. A., Amit, I., Adiconis, X., Fan, L., Raychowdhury, R., Zeng, Q., Chen, Z., Mauceli, E., Hacohen, N., Gnirke, A., Rhind, N., di Palma, F., Birren, B. W., Nusbaum, C., Lindblad-Toh, K., Friedman, N., and Regev, A. (2011). Trinity: reconstructing a full-length transcriptome without a genome from RNA-Seq data. *Nature Biotechnology*, 29(7):644–52.

Gymrek, M., McGuire, A. L., Golan, D., Halperin, E., and Erlich, Y. (2013). Identifying personal genomes by surname inference. *Science*, 339(6117):321–4.

Hillion, K.-H., Kuzmin, I., Khodak, A., Rasche, E., Crusoe, M., Peterson, H., Ison, J., and Ménager, H. (2017). Using bio.tools to generate and annotate workbench tool descriptions. *F1000Research*, 6:2074.

Hosny, A., Vera-Licona, P., Laubenbacher, R., and Favre, T. (2016). AlgoRun, a Docker-based packaging system for platform-agnostic implemented algorithms. *Bioinformatics*, 32(15):2396–2398.

Hunter, A. A., Macgregor, A. B., Szabo, T. O., Wellington, C. A., and Bellgard, M. I. (2012). Yabi: An online research environment for grid, high performance and cloud computing. *Source Code for Biology and Medicine*, 7:1.

Marçais, G. and Kingsford, C. (2011). A fast, lock-free approach for efficient parallel counting of occurrences of k-mers. *Bioinformatics (Oxford, England)*, 27(6):764–70.

Pavelin, K., Cham, J. A., de Matos, P., Brooksbank, C., Cameron, G., and Steinbeck, C. (2012). Bioinformatics meets user-centred design: a perspective. *PLoS Computational Biology*, 8(7):e1002554.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018

Pertea, M., Kim, D., Pertea, G. M., Leek, J. T., and Salzberg, S. L. (2016). Transcript-level expression analysis of RNA-seq experiments with HISAT, StringTie and Ballgown. *Nature Protocols*, 11(9):1650–67.

Schadt, E. E. (2012). The changing privacy landscape in the era of big data. *Molecular Systems Biology*, 8(1):612.

Smith, D. R. (2013). The battle for user-friendly bioinformatics. *frontiers in Genetics*, 4:187.

Smith, D. R. (2015). Buying in to bioinformatics: an introduction to commercial sequence analysis software. *Briefings in Bioinformatics*, 16(4):700–9.

Visne, I., Dilaveroglu, E., Vierlinger, K., Lauss, M., Yildiz, A., Weinhaeusel, A., Noehammer, C., Leisch, F., and Kriegner, A. (2009). RGG: a general GUI Framework for R scripts. *BMC Bioinformatics*, 10(1):74.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018

A APPENDIX

A.1 Use-cases

Life scientists

Many life scientists work in a small wetlab without any significant IT support. The computers in their labs mostly run Windows and PhD students often have to bring their own devices (because the institute does not provide such working devices). Installing major software on the lab computers is a problem, because the administrators have little time to deploy new software.

Oxford Nanopore Sequencing is becoming more and more popular, and even the sequencing hardware can be found in more and more biological laboratories. Particularly important for MinION-sequencing is the post-processing of the actual raw read data. While in previous versions, base-calling has been directly performed in the cloud by Oxford Nanopore, this has now been pushed back to the client side. Thus, despite having the sequencing data on their laptop, scientists must still retrieve the sequences themselves, using, for instance, the Albacore basecaller. Unfortunately, like many bioinformatics packages, the basecaller only comes as a python command-line program. Additionally, the download is only available as a python-wheel, which means there is no regular setup available. The scientists thus need assistance for the installation of the python-wheel as well as starting the basecalling process. After the reads are basecalled, reads need to be aligned to a reference genome. While there exist reference genomes in correct format on the lab computer, the CL program to map the reads is available only from GitHub to be installed from source.

Often a custom analysis of data is not required for an initial screening of the acquired data, but a simple, basic and robust analysis is wanted. Additionally, since life scientists are mostly busy in the lab, an analysis has to be prepared fast, and parameters should be stored for later reference. For this a local searchable database of saved templates is needed.

Software developers

A developer finished his sequence alignment program. The project is already published on GitHub and in a journal, but only few people start using it. From the issues and feature requests on GitHub it can be seen that mainly other bioinformaticians use his program. Thus the developer decides that the program should be usable by more researchers and looks for ways to make his program useable by everyone. Since the program is written in python, it is cross-platform compatible per-se. However, it is noticed that domain-experts fail to install and use the program. Thus the developer must look for an easy way to distribute the software and make it accessible to life scientists. The developer's time is limited, having other projects waiting and gets little support for developing GUI from colleagues, as they have different views on the extent of autonomy a biologist should have regarding sequencing analysis.

A.2 Extending templates with script nodes

Often it is required to perform string-manipulations (e.g. remove file extensions) for command-line arguments. For instance the example below takes as input a HISAT2 index file, and removes the file extension, such that the index will be accepted by HISAT2. For evaluation of this node, the evaluate-function is called with the argv-references are input parameters. The last return-value of the script-callstack is taken as node-output-value.

Listing 1. bioGUI script node with LUA function example. Upon evaluating this node, the evaluate function will be called with the arguments listed in the argv attribute of the script node.

```
<script id="hisat_index_rel" argv="${hisat_index_rel_raw}">
<![CDATA[
function evaluate(arg1)

      if (string.match(arg1, ".%d.ht2$")) then
            return(string.sub(arg1, 0, arg1:find(".%d.ht2$")-1))
      end
   return(arg1)
end
]]>
</script>
```


PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018

A.3 Evaluating a bioGUI template

In Figure 3 the process of assembling a command-line call from the shown *bioGUI* template is explained. First, the creation of the `<window>` model (green) will be explained, following the creation of the command-line arguments using the `<execution>` model (red).

The window component consists of four different components, which are grouped in a vertical layout (default for window component). A label describing the input file dialog is placed on the main window, followed by the actual file dialog with ID `input`. Then a group box with title and a checkable status is created, which contains an output file dialog. Finally, the action button which would start the program and the text output elements are created.

When the user has entered all desired data, and clicks the action button, the execution phase defined by the exection model will be launched. Therefore the program defined in the execute element is started. For this, the parameters (param) must be assembled. Any text within `${var}` is interpreted as a reference to a variable *var* or the value of a GUI element with id *var*. Thus, the command-line is successively assembled. At first the `${input}` element is interpreted and retrieves the value from the input file dialog as this element matches the id. Next the `${output}` is interpreted. The `${output}` refers to an *if* construct in the execution part, which compares the value of the element with id *os* to the string *TRUE* (which is whether the groupbox is checked). If this value is true, this node evaluates to `netcat 192.168.1.100 55025`, otherwise to `tee -a {output file path}`. Finally, the program *sh* is executed with the created command-line arguments. For instance, if the group box is checked, the `sh -c "cat inputFile | netcat 192.168.1.100 55025"` will be executed. A full reference of all input types as well as all execution nodes is available online.

PeerJ Preprints | https://doi.org/10.7287/peerj.preprints.27095v1 | CC BY 4.0 Open Access | rec: 6 Aug 2018, publ: 6 Aug 2018