Continuous integration, delivery, and deployment for scientific workflows in Orlando Tools

S A Gorsky[1]

[1]Matrosov Institute for System Dynamics and Control Theory of SB RAS, Lermontov St. 134, Irkutsk, Russia, 664033

gorsky@icc.ru

Abstract. The paper addresses issues of continuous integration in the development of scientific applications based on workflows (special case of distributed applied software packages) for heterogeneous computing environments. The preparation and carrying out of scientific computational experiments are often accompanied by intensive software modification. Thus, there is a need for the following stages: building, testing, debugging, and installation new versions of software in heterogeneous nodes of environment. These stages may take longer time overheads than computations themselves. The solution to this challenge lies in the use of tools for continuous integration of software. However, such tools require deep integration with the tools for the workflow development because of scientific workflow specifics. To this end, the paper describes the combination of the author's Orlando Tools framework for the development and use packages with the GitLab system that is widely used for continuous integration. Such combination significantly reduces the complexity of software continuous when developing and using packages.

1. Introduction

The increased complexity of computational experiments carried out during scientific experiments requires applying high-performance computing resources. Often, scientific applications are developed in the form of workflows [1], which are similar to problem-solving schemes in applied software packages. A package is a set of programs (modules) designed to solve a specific class of problems in a concrete subject domain. The rapid development of distributed computing technologies has led to significant changes in the package architecture. Packages retained a modular structure. However, it is becoming distributed and oriented towards heterogeneous distributed computing environments. Thus, we have received distributed applied software packages.

Packages have module structures. Each module has a set of various interfaces. The development of each module should be carried out independently from other modules to simplify version control and facilitate team development. To automate software development, it is common practice to use Git, GitLab, and other tools that implement an approach named Continuous Integration (CI). Herein, CI is the software development practice that is based on the merging software working copies to a shared software mainline. Often, CI intersects with additional activities for Delivery and Deployment of software. In such cases, they are united into the uninterrupted pipeline of Continuous Integration, Delivery, and Deployment (CI/DD).

The CI/DD tools are used in both the development of fairly simple programs or module libraries by various programming languages and design of complex software systems. The latter case relates to the

considered packages. The use of continuous integration tools in developing packages is fraught with a wide range of difficulties. The main one related to the fact that the modules of a workflow (problem-solving scheme) can be placed into several Git repositories. As a rule, well-known continuous integration systems do not fully support control of software modifications for hierarchical repositories.

This paper proposes the solutions used to automate the development and modification of distributed applied software packages in the Orlando Tools framework [2]. These solutions significantly reduce both the overheads for the package development and time spent on computing.

The rest of the paper is organized as follows: the next section briefly reviews related works. Section 3 describes the organization of continuous integration in Orlando Tools. Section 4 illustrates the capabilities of Orlando Tools using the example of Grad2 package. Section 5 provides an example of the implementation of a continuous integration scheme for the Grad2 package. Section 6 concludes our study.

2. Related work

There are the following main stages of the development and use of a scientific workflow within a distributed applied software package:

- Development and modification of modules,
- Testing of modules,
- Development and modification of the workflow description,
- Installation of modules on computing resources,
- Testing of workflow,
- Computations.

From this point of view, the workflow development process does not look complicated. However, a detailed analysis of the process shows that each of these stages internally hides many problems. In addition, these steps are interrelated and are often multiple repeated.

Any modification of the workflow module entails its testing and installation on computing resources. Modification of the description of resources, workflows, and modules related them may also be required. For example, changes in computing resources, such as the appearance of a new resource or a change in resource configuration, may require adaptation of modules, their repeated installation, and testing. Identifying problems within the workflow testing can also entail repeating the aforementioned stages. The solution of these problems lies in the field of CI/DD of software.

There are two scenarios to integrate workflow development tools and CI/DD tools. The first scenario assumes, that developers use common practice for the software development process. Within this scenario, software development is under the control of CI/DD tools. It is based on the use of repositories. Therefore, it is necessary that the workflow management system has a set of APIs for the interaction with the CI/DD tools and enables to describe a workflow in a text form. An analysis the well-known universal CI/DD tools (for example, CircleCI [3], Jenkins [4], TeamCity [5], Travis [6], GitLab [7], etc. [8, 9]) and workflow development tools have been made. Based on this analysis, it is obvious that developers are faced with significantly complicated problems in workflow development process, including the following:

- There is no information in the workflow about the repositories used for its module,
- Developers need to use various software tools that are not united by common interfaces and relations between the controlled objects,
- Information about resources is separated between the workflow description and module repositories.

In the paper, it does not claim that a developer cannot automate the implementation of its individual stages for a specific workflow. Nonetheless, the integration of workflow development tools with CI tools often raises the requirements for workflow developers and lies to great time overheads.

The second scenario assumes, that the user may access the CI/DD capabilities as part of the workflow development tool. Such a scenario is implemented in Orlando Tools. Within the represented overview of the related work, systems that fully implement this scenario have not been found.

3. CI/DD in Orlando Tools

Orlando Tools is aimed at the development of distributed applied software packages for large-scale computing experiments. It supports the design and application of workflows in a heterogeneous distributed computing environment.

The main Orlando Tools subsystems are the user interface, conceptual model designer, software manager, and computation manager. They were considered in detail in [10].

Orlando Tools is installed on a dedicated Linux machine. Currently, virtual machines with Ubuntu 18 are used. End-users work with Orlando Tools through a web interface.

3.1. Integration with GitLab

As a result of the comprehensive analysis of the CI/DD tools [2], the GitLab system was selected as the external CI/DD system for the combination with Orlando Tools. This system uses the capabilities of Git and specialized APIs that provide access to the repositories themselves and CI/DD functions. The combination with GitLab is carried out on two levels.

The first one is the programming level related to the package modules development. The module development is carried out using the standard approach. It includes the following operations: module code design or modification, executable module version build, its testing and placing on computing resources. Any workflow management systems can be extended with such programming level using external CI/DD tools through creating additional software to interact with these tools. In comparison with such systems, Orlando Tools provides tools for working with Git and GitLab repositories. These tools are built into the software manager of Orlando Tools. Among them are a built-in editor and monitoring system. The built-in editor allows us to work with module repositories, build repository dependency trees, and configure continuous integration in GitLab. The monitoring system supports the run of continuous integration chains within both the module repositories and the entire package.

The second level is implemented as part of the conceptual model designer. Within this level, package descriptions and all related information are stored and processed in the GitLab repository. Thus, all package elements are reflected in the GitLab repositories. This provides developers with access to the GitLab capabilities at both the module level and package level.

The GitLab application is optional for the Orlando Tools end-users. The decision for each module can be made independently of the other package modules. Developers make decisions for selecting the required automation level for the compilation, build, testing, delivery, and deployment stages of software developing process.

The maximum functionality involves storing the module source code on the GitLab server and automating all above-listed processes. Within the partial functionality, only the module description can be stored in the GitLab repository for its use during package development. Without GitLab, all information about modules are contained in the package description.

3.2. Module deployment

An important feature of Orlando Tools is the automation of the module installations on the environment resources. Module installations can be accompanied by the module testing on these resources. This is especially relevant with frequent modifications of package modules and its descriptions.

3.2.1. Creation of heterogeneous distributed computing environments

All information about the computing environment is centralized and stored on the Orlando Tools server. The environment includes different heterogeneous resources, packages and repositories. Information about the computing resource includes the following characteristics:
- Recourse name used to operate with the resource within the package,

- System tags that identify the system characteristics of the resource (operating system, type of computing resource, etc.) and allows to select options for installing modules,
- Resource tags that identify the role of the computing resource and allows to determine which modules should be installed on the resource,
- Network address that provides access to the resource via the SSH protocol,
- SSH port,
- User login,
- User password,
- Directory of scripts that implement the Orlando Tools interface with the resource,
- Directory of temporary files used to store temporary files of tasks launched on a resource,
- Maximum number of nodes for the task,
- Number of cores on nodes,
- Maximum number of tasks per user in the task queue,
- Maximum number of cores per task.

Access to environment resources is carried out under the SSH protocol. To unify operation with resources, a set of the following scripts is used:
- Script for obtaining information about the resource status,
- Script for launching tasks,
- Script for checking the task status,
- Script for deleting tasks.

In practice, adaptation these scripts allows us to connect almost any computing resource to the Orlando Tools via the SSH protocol, including resources running under OS Windows.

3.3. Continuous integration at the package level

The CI/DD process in Orlando Tools can be divided into the following stages:
1. Preparation of executable module versions in GitLab. Wherein, Orlando Tools can respond to changes in the repository tree and run the pipeline in the top-level repositories.
2. Parsing the package description to take into account new possible changes after the completion of all CI/DD processes at the GitLab level. In the process of parsing, resources for the installations and testing modules are determined.
3. Installation and testing modules on test resources. Each module is installed on only one resource. If no resource is marked as a test one, then the module is installed on the first available resource. If the necessary settings are made, then the module testing is started immediately.
4. Testing workflows. In the case of successful testing modules, all workflow tests are launched.
5. Installation of modules on the remaining marked resources with subsequent testing.
6. Completion of CI/DD.

3.4. Repository monitoring

Orlando Tools supports monitoring package repositories. Such monitoring provides the pipeline run for repositories related to the already modified repositories. For example, Figure 1 shows that a change in the lib1.master repository will cause the pipeline to run in the lib1.master repository. Therefore, changes can spread from the leaves of the repositories tree to its root. This allows us to avoid the GitLab free version limitation, which does not support this feature.
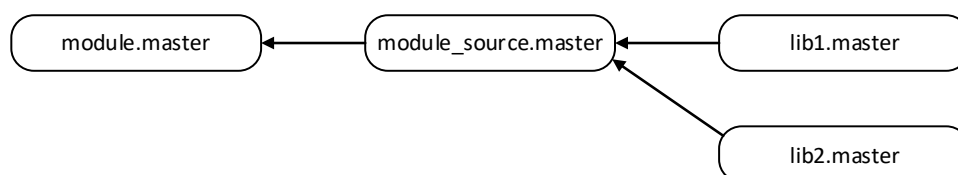

Figure 1. Repositories tree of the module *gradient*.

The module repository is set in the package description. The description of the rest of the repositories tree is stored in the file *repos_tree.json* of the module repository. Figure 2 demonstrates the content of this file for the repository *module.master*.

```
{"repos":[
 {     "sever": "gitorlando",
       "project": "example/module_source.master",
       "repos":[
               {     "sever": "gitorlando",
                     "project": "example/lib1.master",
                     "after": "startparent"
               },
               {     "sever": "gitorlando",
                     "project": "example/lib2.master",
                     "after": "startparent"
               }
        ]
}]}
```
Figure 2. Content of the file repos_tree.json.

3.5. Module deployment methods

The module installation on resources of a heterogeneous distributed computing environment taking into account the intensive module modification takes a lot of time. With the limited number of resources, the automation of the module installation using external CD tools is possible. With a large number of resources, it is very difficult.

To this end, Orlando Tools supports three methods of the module installation on environment resources.

The first option involves the manual module installation on resources. It is the most time-consuming. In this case, information about the location of the module installation can be included in the package description or placed to the file *orlando_res.json* in the module repository. This option may be relevant for rarely modified modules (for example, for licensed programs tied to the concrete resources).

The second option involves installing modules on static resources. It is implemented using the GitLab tools by adding installation actions to a CI/DD pipeline. In the example in Figure 1, these actions can be included in the pipeline for the repository *module_source.master*. Information about the resources on which the module is installed must be written in the file *orlando_res.json* of the repository *module.master* or be added in the package description.

The third option is intended to automate the process of installing modules on resources. It involves the use of the Orlando Tools capabilities. In this case, the module must have a GitLab repository that stores information about the module, data for its installation, and data for the module testing. In the example in Figure 1, this is the repository *module.master*.

Various software and hardware of resources makes it necessary to support multivariate installation of modules. To implement this feature, Orlando Tools uses resource tags. Each resource has a set of tags. Various scripts are used for various module installation options. All information about the installation options for the module is written the file *orlando_install.json*. It looks as follows:

```
{"installs":  [{"tags": "linux","install": "./install.sh"}]}
```

The file *resources_tag.json* contains the resource tags on which the module must be installed. The additional field *test* indicates the resources used to test workflows of the package. The file *resources_tag.json* contains the following information:

```
{"tags":
    [
        {"tag" : "cluster"},
        {"tag" : "node", "test" : "true"}
    ]
}
```

The algorithm of the resource determination for the module installation includes the following stages:
1. Orlando Tools processes the module repository and reads the files *resources_tag.json* and *orlando_install.json* in parsing the package description.
2. Orlando Tools searches for resources marked with the read tags. A resource must contain at least one module tag.
3. In the database, Orlando Tools marks the resources on which the modules should be installed. Commands for installing the module on these resources and testing it are also stored in the database.
4. Orlando Tools processes the file *orlando_res.json* and store information from it to the database.

As a result of this algorithm, the database stores the resources on which the module is installed, and the resources on which it is necessary to install it. Additionally, all the necessary information is added to the database for the subsequent stages of installing the module on computing resources and testing it.

3.6. Module installation on resources

Initially, Orlando Tools tries to ensure the module installation on one of the test resources. If there are no available test resources, then an attempt is made to install on any available resource associated with the module. To install the module on the resource, the following steps are performed:
1. Connection to the resource via SSH, creation a temporary installation folder, cloning of the repository, and installation command run.
2. Getting the path to the module executable file from the installation script and writing this information into the database.
3. If a testing command is stored in the database, then Orlando Tools starts the module test. Otherwise, it is marked that the module is installed.
4. If the module is successfully tested, Orlando Tools is marked that the module is tested. Otherwise, the error message is sent to the developer.

This algorithm guarantees the availability of resources for the modules during testing process of the package computational schemes.

3.7. Module testing

Untested modules can be used during computation. However, the system will give preference to resources on which the module has already been tested.

In case of successful module execution, a mark about the module checking on the resource is stored in the database. This mark is also set for the module that is a part of the successfully executed workflow of the package on the resource.

Input test files for the module and a test script should be placed in repository. CI/DD in Orlando Tools involves running a test script for the installed module on the resource. Test run commands are also stored in the file *orlando_install.json*:

```
{"installs":  [{"tags": "linux","run": "./install.sh", "test":"./test.sh"}]}
```

A completion code of 0 should be returned if successful. According to the test results, relevant information is stored into the database.

4. Practical application

Illustrative examples of operating with repositories are based on the *Grad2* package. This package is used to search for global minima of complex multi-extreme functions using the multi-start method. The development and use of this package are provided in detail in [11].

A description of the package is given below in the specialized language CDML (Listing 1). The description in XML can also be used. However, it is more complicated.

```
1.    Project grad2
2.    Parameter Function txt;
3.    Parameter Params txt;
4.    Parameter StartPoints txt [ArrayCount_0_s];
5.    Parameter EndPoints txt [ArrayCount_0_s];
6.    Parameter PointCount txt;
7.    Parameter Result txt;
8.    Parameter Dimensions txt;
9.    Parameter Limits txt;
10.   Parameter ArrayCount txt;
11.   Module    generate    (Dimensions,Limits,PointCount,Function,Params>
      ArrayCount,StartPoints[]) gitorlando grad2/gen.master;//gen %%3 %%1 %7
      %2 %%6 %4 %5
12.   Module    gradient    (Dimensions,Function,StartPoints,Params>EPoints)
      gitorlando grad2/grad.master;//grad %%1 %3 %5 %4 %2
13.   Module  complete  (Dimension,EndPoint[],ArrayCount>Result)   gitorlando
      grad2/res.master;//res %%1 %2 %%3 %4
14.   Operation  Generate  module  generate  (Dimensions,Limits,PointCount,
      Function,Params>ArrayCount,StartPoints);
15.   Operation Gradient module gradient (Dimensions,Function,StartPoints,
      Params>EndPoints) [ArrayCount_0_s];
16.   Operation   Complete   module   complete   (Dimensions,EndPoints,
      ArrayCount>Result);
17.   Task grad (Dimensions,Limits,PointCount,Function,Params>Result);
```
<center>Listing 1: Package description.</center>

This package consists of one project, which includes three operations and three modules that implement the operations. The task *grad* describes traditional parameter sweep computations. It searches global minima (the parameter *Result*) by parameters values: *Dimensions, Limits, PointCount, Function,* and *Params*. These parameters store: the dimension of the space, the search area, the number of starting points, the function under study, and configure the local descent. The intermediate parameter *StartPoints* stores the starting points values divided into several arrays processed independently by instances of the operation *Gradient*. The result of each instance of the operation *Gradient* is the smallest local minima from its results. The operation *Complete* selects the smallest one from the found local minima and is taken as global minima.

A graphical representation of the workflow is shown in Figure 3. On the left part of the screenshot, we can see the tree of the package objects. Specialized web-forms are used to create and modify the objects. After completing all the changes, a package description in CDML or XML is generated in the automated mode.
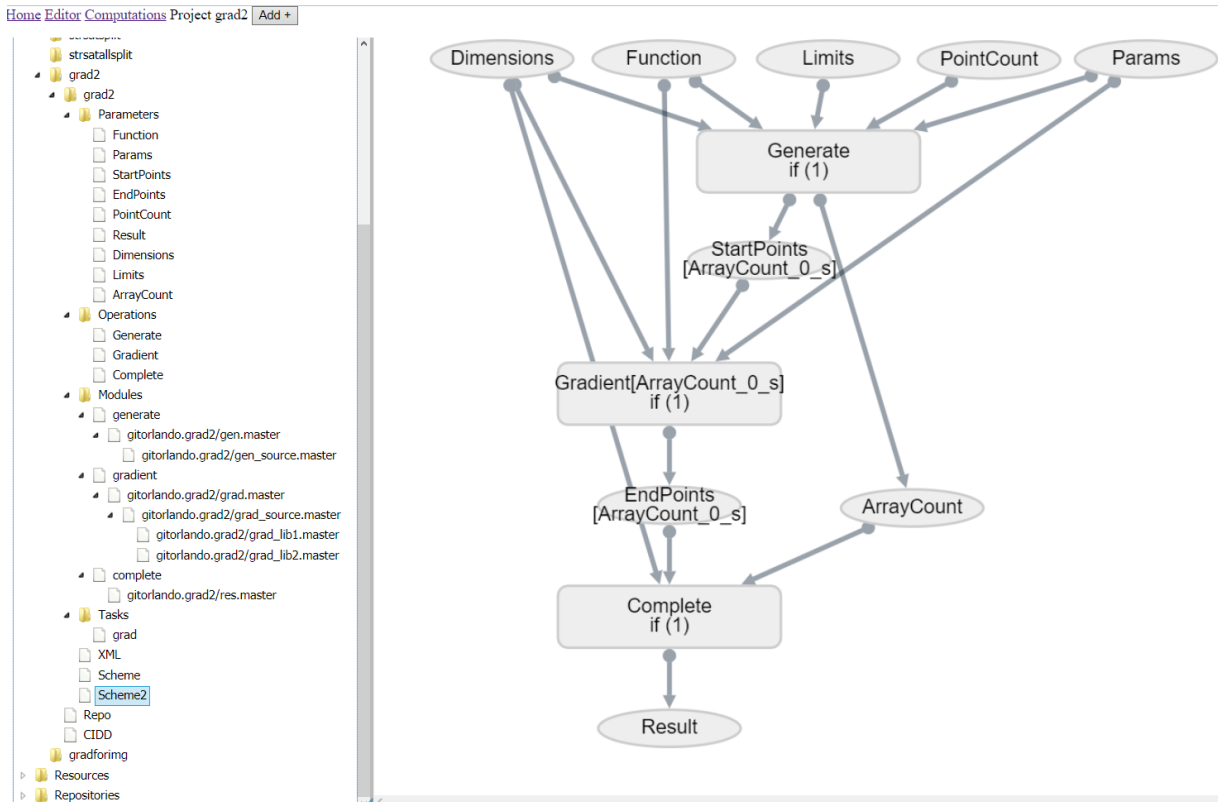
Figure 3. Graphical representation of the package in Orlando Tools.

On the screenshot, the repository trees for the modules are also displayed. They allow us to add, delete, edit, and configure repositories dependencies.

The screenshot in Figure 4 shows the tools for editing the source code of the modules and configuring CI/DD. File manager elFinder [12] is used to operate with the repository. This screenshot reflects the process of editing the description of the *Grad2* package and uploading its new version to the repository. At the bottom of the screenshot, the fragments of messages displayed in the log file show the sequence of operations performed in Orlando Tools.
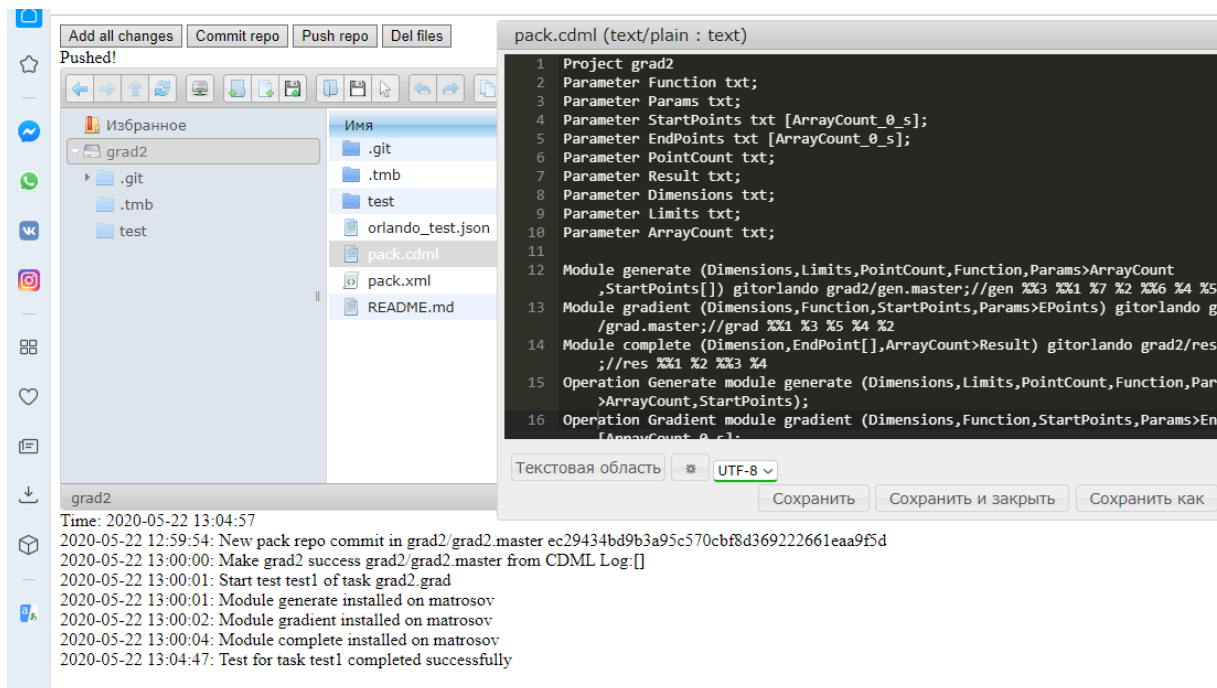
Figure 4. Orlando Tools repository editor.

5. Example of the CI/DD implementation

As an example, let us consider making changes to the library *grad2/grad_lib1.master*. This library provides the calculation of a mathematical function specified in the symbolic form. In this regard, the calculation of a function is very time-consuming. It significantly affects workflow execution time.

A set of user functions is defined in the same library. Accordingly, this library often changes as new learning functions are added. The process of implementing CI/DD at changing this library includes the following stages:

1. After making changes to the source code of this library and saving the new version in the repository, the GitLab server launches the pipeline described in the file *.gitlab-ci.yml*. In this package, one compilation server is used to compile all programs. All libraries are stored on the server in a shared folder.

2. Orlando Tools monitors the status of package repositories in which the CI/DD function is activated. After receiving information that a change has occurred in the repository grad2/grad_lib2.master, the pipeline result of this repository is checked and the repositories depending on it are determined.

3. There are two depended repositories *grad2/gen_source.master* and *grad2/grad_source.master*. Thus, corresponding messages are sent to the GitLab server to run pipelines in these repositories. This information is stored into the database.

4. After the completion of pipelines in these repositories, repositories of the modules *generate* and *gradient* are updated (executable files are updated).

5. Making sure that all previous actions are completed successfully, Orlando starts a parsing process of the package description stored in its repository.

6. In the process of parsing, the repositories of modules are processed. Next, the installation of modules on resources is initiated.

7. The software manager installs the modules on resources.

8. The CI/DD system tests the workflow *grad* on test data.

Throughout the CI/DD process, the developer has access to the current report about its fulfilment. In particular, if errors occurred during the CI/DD process at any its stage, the developer will receive corresponding information. Figure 5 shows the compilation error in the package module caused by the changes in the library in another repository. At the same time, the developer is provided with comprehensive information necessary to correct errors.
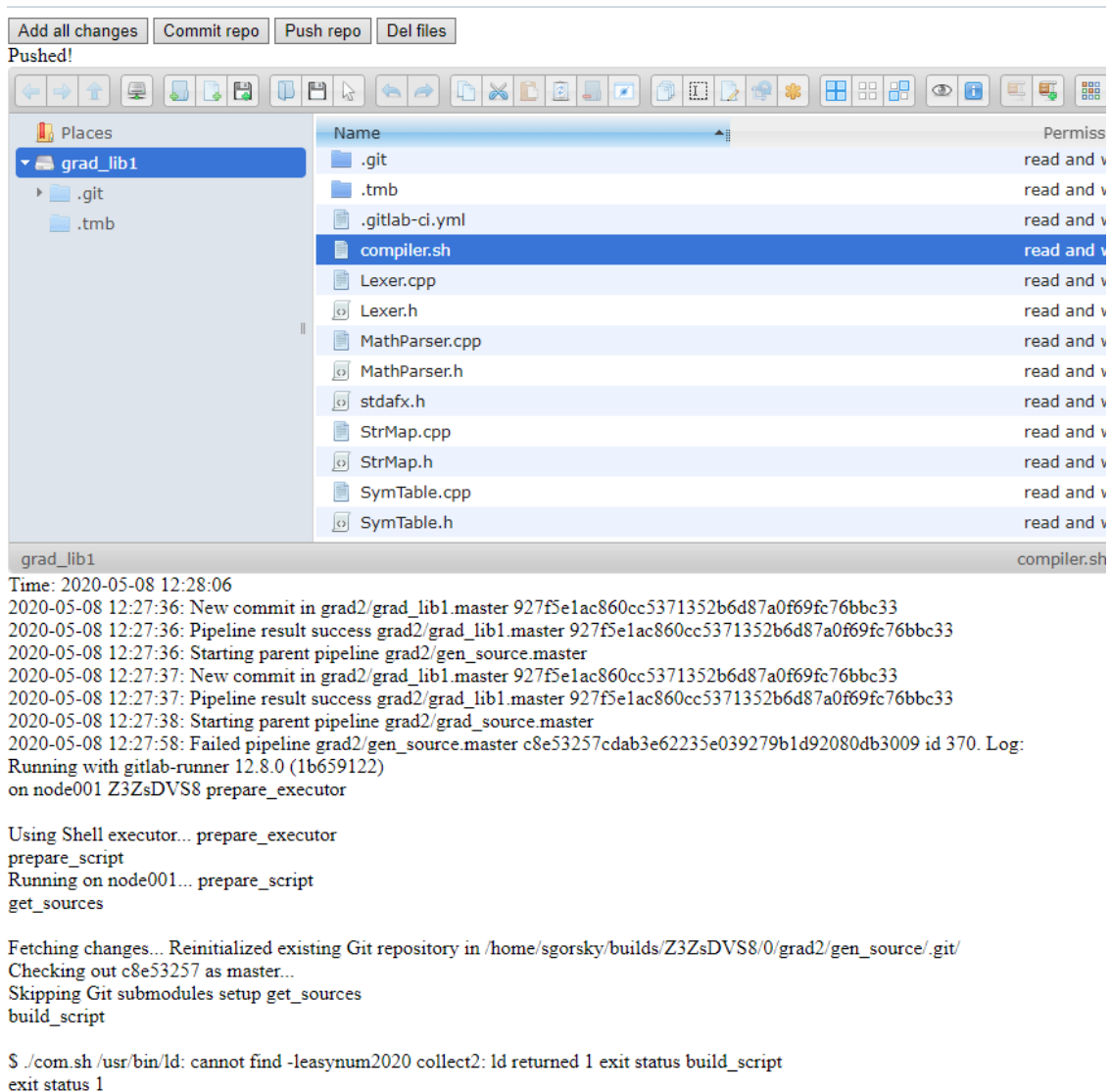

Figure 5. Example of the CI/DD log.

6. Conclusions

In the paper, it is proposed a new scheme for supporting the continuous integration of distributed applied software packages based on the workflows (problem-solving schemes) used. Applying the proposed scheme provides a significant reduction in overheads for continuous software integration. Working with the package as a single object and the integration of Orlando Tools with GitLab allowed users to quickly access all repositories and the results of continuous integration. Automating processes of the package

deployment and testing on heterogeneous resources of a computing environment increases the computing reliability and reduces the time spent for large-scale experiments in whole.

Future plans of a study are oriented to expanding the capabilities of Orlando Tools in relation to a partially automation of the resource configuration in GitLab within of CI/DD. In addition, it is supposed to expand the ability to manage branches and tags within several repositories. It will automate the process of creating new branches within the package development.

References
[1]   Barker A 2007 Scientific workflow: a survey and research directions *Lect. Not. in Com. Sc.* **4967** 746–53
[2]   Feoktistov A, Gorsky S, Sidorov I, Bychkov I, Tchernykh A and Edelev A 2020 Collaborative Development and Use of Scientific Applications in Orlando Tools: Integration, Delivery, and Deployment *Comm. Com. Inf. Sc.* **1087** 18–32
[3]   Sochat V 2018 Containershare: Open Source Registry to build, test, deploy with CircleCI *J. of Open Sour. Soft.* **3(28)** 1–3
[4]   Soni M and Berg A M 2017 Jenkins 2.x Continuous Integration *Packt Pub.* p 438
[5]   Machiraju S and Gaurav S 2018 Deployment via TeamCity and Octopus Deploy *DevOps for Azure App.* 11–38
[6]   Beller M, Gousios G and Zaidman A 2017 Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub *Proc. of the 14th Int. Conf. on Mining Soft. Rep.* 356–67
[7]   Gruver G 2016 *Start and Scaling Devops in the Enterprise* (BookBaby) p 100
[8]   Shahin M, Babar M A and Zhu L 2017 Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices *IEEE Access* **5** 3909–3943
[9]   Wolff E 2017 *A Practical Guide to Continuous Delivery* (Addison-Wesley) p 265
[10]   Tchernykh A, Bychkov I, Feoktistov A, Gorsky S, Sidorov I, Kostromin R, Edelev A, Zorkalzev V and Avetisyan A. 2020 Mitigating Uncertainty in Developing and Applying Scientific Applications in an Integrated Computing Environment *Prog. Comp. Soft* in press
[11]   Bychkov I V, Oparin G A, Tchernykh A N, Feoktistov A G, Gorsky S A and Rivera-Rodriguez R 2018 Scalable Application for the Search of Global Minima of Multiextremal Functions *Optoel., Instrum. and Data Proc.* **54(1)** 83–89.
[12]   https://github.com/Studio-42/elFinder