



Moving to Gitlab issues

1	Contents	
2	Current status	2
3	Organization	3
4	Fields in Phabricator	3
5	Templates in Phabricator	3
6	QA Report App	3
7	Apertimes	4
8	Workflow with Phabricator	4
9	Proposal	5
10	Fields in Gitlab	6
11	Templates in Gitlab	7
12	Permissions	7
13	Management data and view	7
14	Workflow	8
15	Summary	8
16	Export/import	9
17	Migration	9
18	Steps	10
19	Apertis is an Open Source project which has been growing sustainable during	
20	the past years. This growth also made it spread across different projects and	
21	teams, requiring Apertis to improve the tools it uses. In this regard, one issue	
22	that prevents Apertis to be really open is the fact that the bug tracking system	
23	is only open to maintainers making it hard for the community to report new	
24	bugs or to keep track of them.	
25	Since everyone can have access to Gitlab, the most reasonable approach would	
26	be to use Gitlab issues for the bug tracking. The current document describes	
27	the plan for the migration.	

28 Current status

29 Nowadays Apertis uses [Phabricator](https://phabricator.apertis.org)¹ to track bugs but it is only accessible
30 to users with privileges, such as maintainers or developers. One additional
31 drawback is that Phabricator is an open source tool which is [no longer supported](https://www.phacility.com/phabricator/)
32 [upstream](https://www.phacility.com/phabricator/)².

¹<https://phabricator.apertis.org>

²<https://www.phacility.com/phabricator/>

33 Organization

34 Currently bugs are treated as Phabricator **tasks** with a bug related tag such as
35 **bug**, **bug (warranty)**, **bug (not-warranty)**. The meaning of these tags are
36 only important for the management team inside Apertis, and has no value for
37 the community.

38 Bugs can also have additional tags which allows:

- 39 • Release: Bugs can be tagged with a release such as **v2021** or **v2022** to
40 describe that the issue is present in specific releases.
- 41 • Test-failure: This tag is used to denote bugs that are reported during the
42 QA process, either from an automated test in LAVA or from a manual
43 test.
- 44 • Topic: Additional tags such as **infrastructure**, **licensing** can be added
45 to allow easy filtering of possible connected issues.

46 The current workflow relies mostly on the warranty/not-warranty, test-failure
47 and on the release tags.

48 Fields in Phabricator

49 Phabricator supports special fields that help both developers and the manage-
50 ment team to keep track of the progress. The list of fields are:

- 51 • Priority: Defines a priority for the issue, possible values: **highest**, **high**,
52 **needs triage**, **normal**, **low**, default value **needs triage**
- 53 • Status: Keeps track of the progress of the work, possible values: **un-**
54 **confirmed**, **confirmed**, **in progress**, **proposed**, **submitted**, **resolved**,
55 **verified**, **closed**, **wont fix**, default value **unconfirmed**
- 56 • Visible to: Used to configure some tasks only visible to a group of people
- 57 • Tags: Project defined tags
- 58 • Subscribers: List of people that get a notification on changes, members of
59 the management team.

60 Templates in Phabricator

61 Currently Phabricator presents a pre-formated [bug form](#)³ page to create a new
62 bug task. This helps developers to keep a consistent format across bugs, as well
63 as avoiding missing important information.

64 QA Report App

65 Quality Assurance in Apertis is managed through [QA Report App web inter-](#)
66 [face](#)⁴ which summarizes the report for all the supported releases. These report
67 are the result of running Apertis [test cases](#)⁵ on images either from an automated

³<https://phabricator.apertis.org/maniphest/task/edit/form/8/>

⁴<https://qa.apertis.org/>

⁵<https://qa.apertis.org/>

68 or manual test.

69 Automated tests are run on image creation using LAVA and results are reported
70 back to QA Report App. On test failures this application automatically gener-
71 ates either a new bug or append new entries if a previous open bug for the same
72 test case is found.

73 Since some test cannot be completely automated a set of them are run manually.
74 The results of these tests are submitted to QA Report App in order to have a
75 common source of information. In case of failure, following the same approach
76 as for the automated tests, either a new bug or a new comment in an open bug
77 is introduced.

78 Apertimes

79 Management team inside Apertis uses **apertimes** to track progress of the
80 project. This tool retrieves task information from Phabricator to generate re-
81 ports.

82 Workflow with Phabricator

83 The current workflow used for bug tracking can be described through these
84 steps:

- 85 • Bug report: A Phabricator task is created when a new bug is found, there
86 can be three situations:
 - 87 – Manual report base on user work: In some cases a user performing
88 some kind of work might notice some unexpected behavior. In this
89 case the user is encourage to create a new bug using the [bug form](#)⁶.
 - 90 – Manual report based on test suite: Apertis provides a [test suite](#)⁷ for
91 its releases, some of which need to be run manually. When the QA
92 process is run a failure is reported through the [bug form](#)⁸.
 - 93 – Automatic report based on LAVA jobs: As mentioned the [test suite](#)⁹
94 includes automated tests that are run on LAVA. **QA Report App**
95 automatically creates a bug on failures.

96 On bug reporting it is important to try to include as much accurate information
97 as possible. In this regard, a proper title, description and tags can help on later
98 steps.

- 99 • Bug triage: On weekly basis the list of bugs is checked and new bugs are
100 triaged with a priority as mentioned in **Fields in Phabricator**. Additionally,
101 during this process for bugs with **highest** or **high** priority that require
102 urgent attention, a developer is assigned. In other cases, developers claim a

⁶<https://phabricator.apertis.org/maniphest/task/edit/form/8/>

⁷<https://qa.apertis.org/>

⁸<https://phabricator.apertis.org/maniphest/task/edit/form/8/>

⁹<https://qa.apertis.org/>

bug based on the priorities and their skills. Additionally, tags can be added to make it easier to connect related bugs and to help the management team to keep track of the task in progress.

- Once a developer starts to work on a bug they change the **status** to **in progress** and updates the task regularly with the progress.
- After debugging, a solution is designed and proposed, usually by submitting a Draft [Merge Request](#)¹⁰ to discuss the approach. When doing this **status** is updated to **proposed**, to show that there is a proposed possible fix for the issue.
- During the review process different things can happen
 - The proposed solution is accepted by a developer with experience in the field in which case it can be merged. This is usually the case for small fixes.
 - The proposed solution is accepted but additional work needs to be done to fix the issue. This goes from additional work on the MR or a new MR with additional changes. In this case when additional MR are submitted the **status** can be changed to **submitted**.
 - In some cases the root cause of the problem is in an upstream issue, from which information or feedback is required. Under such circumstances, the **status** can be updated to show that the developer is waiting for upstream by setting it as **upstream**.
- When all the MR are merged and no additional work needs to be done the **status** should be updated to **resolved**.
- Finally, the reporter can check if the issue is solved in which case the **status** should be changed to **verified**. If the reporter has doubts about the issue, he can request additional information and set the **status** to **need info**.
- In some cases, after investigating a bug, evaluate its impact and the possible solutions the best action is not to fix it. A good example of this case are packages no longer supported. In order to make this clear the **status** should be updated to **wont fix**.

During all the steps described any developer can add comments to help to fix the issue. It is also possible that members of the management team queries the status of the bugs.

Proposal

As previously mentioned, given all the Apertis related projects are already hosted in Gitlab, the use of Gitlab issues is natural. For this, Gitlab provides

¹⁰https://gitlab.apertis.org/dashboard/merge_requests?scope=all&utf8=%E2%9C%93&state=opened&assignee_id=None

140 a per project issue tracking system which is the best approach to handle de-
141 velopment. Gitlab also support **incidents**, but those are focused on service
142 disruption.

143 However, since Apertis is an Open Source Distribution, the mindset is different,
144 each individual upstream project usually has its own bug tracking system, while
145 Apertis focus on tracking issues from a distribution's integration point of view.

146 Also, requiring contributors to first find the right project to report the issue
147 will not be user friendly, and as consequence some users will avoid reporting the
148 problem or possibly report it to the wrong project.

149 For these reasons, the recommendation is to create a new single project for bug
150 tracking, which will also help the management team, since there will be a new
151 different issue id for every new issue. This is the same approach Debian uses in
152 its [bug tracking system](#)¹¹.

153 This approach has also the advantage of making transition from Phabricator
154 easier as issues will be migrated to only one project.

155 During the migration open bugs in Phabricator will be added to Gitlab with
156 title, description, priority, status and tags set. In the description, a link to the
157 Phabricator task will be added to more easily check the history. Importing the
158 history is not recommended since there could be sensitive data, additionally
159 comments also wouldn't have a correct datetime set. For closed bugs, since
160 there is no value on importing them without the history the recommendation is
161 to keep them only in Phabricator.

162 After the import, which is one time operation, Gitlab issues will be the main
163 repository for open or new bugs.

164 The use of tags in Phabricator can be emulated in Gitlab with labels, making
165 the transition almost transparent for end users.

166 However the use of tags for private management is discouraged since all informa-
167 tion is made public. This topic will be covered in more detail in the **Management**
168 **data and view** section.

169 Fields in Gitlab

170 Gitlab issues does not provide the same fields, however, similar functionality
171 can be implemented using [labels](#)¹². To do so, for each field a set of labels should
172 be created for every possible value, providing also additional flexibility.

173 For example, to implement the **priority** field, a set of labels **priority: highest**,
174 **priority: high**, **priority: normal** and **priority: low** should be created. The
175 same approach can be used for **status** since it also has a defined set of possible
176 values.

¹¹<https://bugs.debian.org>

¹²<https://docs.gitlab.com/ee/user/project/labels.html>

177 However the visibility field does not make much sense on this approach since
178 the plan behind this task is to openly show the bug tracking system to the
179 community, in the same way other Open Source projects do. In case of an
180 exception, Gitlab supports the use of [confidential issues](#)¹³ which has visibility
181 set to only team members.

182 Tags can be easily implemented by labels since they have the same behavior,
183 providing a very flexible way of connecting issues.

184 Finally, subscribers can be handled using the [Gitlab notifications](#)¹⁴. [Issues](#)
185 [templates](#) can be configured to notify people about new activity by sending
186 an email allowing all team members to be informed. Gitlab supports different
187 [notification levels] providing a way for users to configure them according to their
188 needs. Since the template will issue a mention to a default list of subscribers,
189 everyone in that list who have not disabled notification will receive an e-mail
190 on bug report and in subsequent comments.

191 Templates in Gitlab

192 Gitlab [templates](#)¹⁵ are used to have a common pattern to report bugs. Gitlab
193 supports creating templates using markdown and thus allowing an easy way to
194 import the current Phabricator templates.

195 The management of templates is straightforward since they are file located in
196 the `.gitlab/issue_templates` folder, so the creation, modification and deletion
197 can be handled thorough **merge requests**, which also helps traceability.

198 Permissions

199 As described in [permissions](#)¹⁶, Gitlab supports different roles in a project and
200 since bugs will be a new one, permissions can be easily adjusted as needed.

201 In order to provide public access to reported issues, contributors need to be
202 assigned the role of **guest** to the project. In this way they can create issues but
203 won't be able to change labels or assign them after creation.

204 Management data and view

205 Gitlab issues is a powerful tool to keep track of different tasks, in this case
206 bugs. However, due to the private nature of some aspects of these tasks some
207 information should be placed in a different location.

208 On the public project useful information can be included to help the manage-
209 ment team, as example new labels based on **bug-type** should be used:

¹³https://docs.gitlab.com/ee/user/project/issues/confidential_issues.html

¹⁴<https://docs.gitlab.com/ee/user/profile/notifications.html>

¹⁵https://docs.gitlab.com/ee/user/project/description_templates.html

¹⁶<https://docs.gitlab.com/ee/user/permissions.html>

- `bug-type:apertis`: To denote a bug that appeared in Apertis due to work done inside Apertis project
- `bug-type:upstream`: To denote a bug that appeared in Apertis due to a bug also present upstream

Any private information should be placed in a different location, such as a private GitLab project or an external tool, and the mapping between issues and management data should be performed manually.

The easiest way to implement this is to keep private information in Phabricator as it is the tool currently used, reducing the complexity of the solution. With this approach bugs will be handled on Gitlab issues but a Phabricator task will be created and linked to track private management information. To make this possible a tool should be developed to create a Phabricator task based on a Gitlab issue with the same title and a description, including a link, referencing it. The management team will use this task to track only private data. Lastly, since management data will remain available in Phabricator, **apertimes** tool doesn't require any update, making the transition easier.

For public data, to have a high level overview of bugs, Gitlab support [boards](#)¹⁷ which provide a kanban style interface. Boards columns can be configured based on labels, so the initial approach could be to have a board which shows bugs grouped by priority, which should help tracking the most important issues. It is also possible to create multiple issue boards per project, to highlight other aspects, such as **bug-type**.

Workflow

The current workflow will suffer only minimal changes after switching to Gitlab issues. All the steps described in [Workflow with Phabricator] can be followed using Gitlab issues, however, with the new approach task will be automatically created and linked to Phabricator to allow the management team to hold private data.

Summary

The table below shows a summary for the mapping between Phabricator and Gitlab features/information:

	Phabricator	Gitlab
Organization	Tasks with tag <code>bug</code>	Gitlab issues on new project
Priority field	Internal field	Label with prefix <code>priority</code>
Status field	Internal field	Label with prefix <code>status</code>
Assignee field	Internal field	Internal field

¹⁷https://docs.gitlab.com/ee/user/project/issue_board.html

	Phabricator	Gitlab
Templates	Supported	Supported
Permissions	Supported	Supported
Management view	Workboard	Gitlab boards
Information type	Public and private	Public information only

241 Export/import

242 A key point during the migration is the export and import task, which should
 243 end up with all the open bugs in Phabricator available as Gitlab issues.

244 The desired data to be migrated is:

- 245 • Title
- 246 • Description with a link to the original Phabricator task
- 247 • Priority
- 248 • Status
- 249 • Tags

250 In this regard there are some tools already available such as [import from csv](#)¹⁸
 251 and [import from Phabricator](#)¹⁹. These tools only allow to import title and
 252 description, so importing additional information requires building a tool based
 253 on the Phabricator and Gitlab API. Creating such a tool will likely be rather
 254 simple though.

255 Migration

256 The migration should be planned with enough time before a quarter ends in
 257 order to have a stable system previous to the release dates. It is a pre-requisite
 258 that the **QA Report App** is updated to support Gitlab issues and a set of tools
 259 are developed, one to create and mirror Phabricator tasks based on Gitlab issues,
 260 and another to [Export/import] Phabricator tasks into Gitlab issues.

261 For the actual migration, first a clean up of the current open bugs list should
 262 be performed, to avoid importing useless information. After this point, a freeze
 263 period should be started to guarantee that no modification to the bugs is made
 264 on Phabricator during the migration. This freeze period will be divided in two,
 265 a hard freeze and a soft one.

266 During the hard freeze the export/import operation will be triggered, which
 267 will end with a production test to confirm that all bugs have been migrated and
 268 tools work as expected. The main purpose of this tests is to spot any blocker for
 269 considering Gitlab issues as the main bug repository. For any no-blocker issue
 270 a task will be created. The estimation for this hard freeze period is one day.

¹⁸https://docs.gitlab.com/ee/user/project/issues/csv_import.html

¹⁹<https://docs.gitlab.com/ee/user/project/import/phabricator.html>

271 During the soft freeze period, adding bugs will be enabled only with strict control
272 for just a few people to allow further tests to be run and to fix minor issues.
273 The estimation for this soft freeze period is 3 days. In case that no issues are
274 found the soft freeze period will be reduced.

275 If during the hard freeze period a blocker issue is found which will be unable to
276 fix during the hard freeze, the migration will be aborted and a new migration
277 schedule will be planned.

278 Steps

- 279 • Create test environment on Gitlab
- 280 • Enhance QA Report App to report failure to Gitlab issues instead of
281 Phabricator
- 282 • Create script to export bugs
- 283 • Cleanup bug list
- 284 • Hard freeze bug report/update on Phabricator
- 285 • Export bug list
- 286 • Import bugs list
- 287 • Perform production test
- 288 • Soft freeze bug report/update on Gitlab
- 289 • Additional tests and controlled bug report
- 290 • Unfreeze bug report/update on Gitlab

291 [Workflow with Phabricator] #workflow-with-phabricator