Participant Observation — The How.

Onboarding Event - Details on Technical Activities

Typical Observation study at Berlin Germany, OpenStack Upstream Institute (OUI).

Mentors
: (M1, M2, …, M12)

Participants
: (P1, P2, …, P72), 17 females, 23 neutrals, and 32 males seated on 12 tables forming 12 groups

Outline

Theoretical Knowledge (Day-01) [100%]

- [X] Introduction
- [X] Accounts creation and setup
- [X] Setting up the Development Environment
- [X] How OpenStack is Made
- [X] OpenStack Events

Practical Knowledge (Day-02) [100%]

- [X] Workf-low and Tools for participation
- [X] Code Dive Deep
- [X] Technical Activity

DAY-02 Technical Activities (The "How")

Survey Form

M1 and M2 reminded Mentors to be fully engaged

General Tips for Mentors before day 02 session

1. `Remain engage with the rest of the class even if you are not presenting.`
2. `Choose a table and sit with the students to help`
3. `If there aren't enough mentors or every table has one already, float around the room checking in on people, especially during exercises`
4. `When possible, sit at a table and build connections (networking) with participants`
5. `Talk slowly when you are presenting - English may not be their first language`
6. `Pause to ask students if they have questions on the material throughout your presentation`
7. `Ask the students questions to make sure they are engaged and understand the material`
8. `Join the IRC channel of the class and participate during the training`
9. `Give out swag and make sessions competitive`
10. `Promote ideas for next steps after training is done; mentoring, Project Onboarding and other related conference sessions`

The Lead mentors encouraged participants to be interactive

General Tips for Participants

1. `Use every opportunity you have to give us feedback. It's important for the`

community
2. Discuss your solutions with mentors and explain to them how
   you derived the solutions.
3. Use IRC for answering questions or the training etherpad if an exercise requires more space
4. Be prepared with the "deep dives" exercise, usually, participants have very different levels of knowledge and skillset.

Guidelines

Debugging Code
LOG.debug()

M4 Thought participants techniques how to debug their source code and how debugging applies to Projects.

1. Technical Activities during Day-02

   All 72 participants were actively participating in the coding activities that the mentors assigned them with.

2. Writing and maintaining Quality Code

   1. Git (best practices on git **Commit messages**)

      M7
      gave detailed layout structures of commit messages that respect best practices.

      M7 stated that: "Based on many years of practical experiences doing code development, bug troubleshooting and code review across OpenStack projects and other communities such as Linux kernel, CoreUtils, GNULIB, etc., we suggest a fairly common practice, which is motivated by OpenStack strong desire to improve the quality of it's projects' Git histories."

      M7
      "… We will demonstrate the benefits in splitting up **changes** into a sequence of

      individual commits, and the importance in writing good commit messages to go along with them."

      - M7 divided the topic of commits in two sub topics (A/B), and gave the advantages in splitting commits.
      - The structured set/split of the code changes "If a code change can be split into a sequence of patches/commits, then it should be split."
      - The smaller the amount of code changed, the easier it is to review & identify potential flaws.
      - If a change is found to be flawed later, it may be necessary to revert the broken commit. This is much easier to do if there are not other unrelated code changes entangled with the original commit.
      - When troubleshooting problems using Git's bisect capability, small well defined changes will aid in isolating exactly where the code problem was introduced.
      - When browsing history using Git annotate/blame, small well defined
      - changes aids isolates exactly where & why a piece of code came from.

```
commit 3114a97ba188895daff4a3d337b2c73855d4632d [ID-1]
 Author: [removed]
 Date:   Mon Jun 11 17:16:10 2012 +0100

   Update default policies for KVM guest PIT & RTC timers
```

```
commit 573ada525b8a7384398a8d7d5f094f343555df56 [ID-2]
Author: [removed]
Date:   Tue May 1 17:09:32 2012 +0100

   Add support for configuring libvirt VM clock and timers
```

Furthermore, **M7** explain these two commits, which provide support for configuring the KVM guest timers. The introduction of the new APIs for creating libvirt XML configuration have been clearly separated from the change to the KVM guest creation policy, which uses the new APIs.

1. The information provided in the commit message

M3
: highlight some key points when writing a good commit message

Reflect on these points whenever you are about to write a commit message:

1. Do not assume the reviewer understands what the original problem was.
2. Do not assume the reviewer has access to external web services/site; The commit message should be totally self-contained, to maintain that benefit.
3. Do not assume the code is self-evident/self-documenting.
4. Describe why a change is being made.
5. Read the commit message to see if it hints at improved code structure.
6. Ensure sufficient information to decide whether to review.
7. The first commit line is the most important.
8. Describe any limitations of the current code.
9. Do not include patch set-specific comments.
10. Always use the appropriate tags; Change-id, bug-ID, etc.

Summary of Git commit message structure

- Provide a brief description of the change in the first line.
- Insert a single blank line after the first line.
- Provide a detailed description of the change in the following lines, breaking paragraphs where needed.
- The first line should be limited to 50 characters and should not end with a period.
- Subsequent lines should be wrapped at 72 characters.

```
$ git commit -s [--signed-off-By:]
```

Example of a good commit message shown to participants

```
commit 3114a97ba188895daff4a3d337b2c73855d4632d                    (1)
  Author: [removed]                                                (2)
  Date:   Mon Jun 11 17:16:10 2012 +0100                           (3)
                                                                   (S)
    Update default policies for KVM guest PIT & RTC timers         (4)
                                                                   (S)
    The default policies for the KVM guest PIT and RTC timers      (5)
    are not very good at maintaining reliable time in guest
    operating systems. In particular Windows 7 guests will
    often crash with the default KVM timer policies, and old
    Linux guests will have very bad time drift
                                                                    (S)
    Set the PIT such that missed ticks are injected at the         (6A)
    normal rate, ie they are delayed
                                                                    (S)
    Set the RTC such that missed ticks are injected at a           (6B)
    higher rate to "catch up"
                                                                    (S)
    This corresponds to the following libvirt XML                   (7A)
                                                                    (S)
      <clock offset='utc'>                                          (7B)
        <timer name='pit' tickpolicy='delay'/>
        <timer name='rtc' tickpolicy='catchup'/>
      </clock>
                                                                   (S)
    And the following KVM options                                  (7C)
                                                                   (S)
      -no-kvm-pit-reinjection                                      (7D)
      -rtc base=utc,driftfix=slew                                  (7E)
```

```
                                                                    (S)
    This should provide a default configuration that works          (8)
    acceptably for most OS types. In the future this will
    likely need to be made configurable per-guest OS type.
                                                                    (S)
    Closes-Bug: #1011848                                           (9)
                                                                    (S)
    Change-Id: Iafb0e2192b5f3c05b6395ffdfa14f86a98ce3d1f           (10)
```

- (S) -> White Spaces
- (1) -> Commit ID
- (2) -> Author ID
- (3) -> Date
- (4) -> change request subject (first line of the commit message)
- (5) -> describes the original problem (bad KVM defaults)
- (6) -> describes the functional change being made (the PIT/RTC policies)
- (7) -> describes what the result of the change is (the XML/QEMU args)
- (8) -> describes scope for future improvement (the possible per-OS type config)
- (9) -> uses the Closes-Bug notation
- (10) -> Change-Id
- OB1 :: M3, Made changes to the Sahara project, a cross-projects repository,and explained the changes to participants, then asked participants to write a complete

commit message following the best practices, to describe the changes that were made on the Sahara project. M3: cross"each one of you should discuss with a mentor before submitting the message to the IRC and etherpad doc."

2. Code Quality

   Best Practices
   : Code Quality — Coding Guidelines, Syntax checks and Testing.

   M1 gave a brief explanation on hacking style and listed some advantages "Hacking style guide was enforced by reviewers manually, but the process has been automated. Therefore, hacking makes code written by many different authors easier to read by making the style more uniform. (example: unix vs windows newlines)

   Call out dangerous patterns and avoid them. (example: shadowing built-in or reserved words)."

   M3: gave several use-cases to show how contributors can improve the quality of code by following the Coding Guidelines suggested by OpenStack.

   > (M3) : "to ensure high quality code, OpenStack recommends some syntax checks Frameworks such as: (eslint-config-openstack, Hacking, bashate, etc.), and enforces the OpenStack Coding standard. Experience shows that when contributors write code that respect the Coding Guidelines proposed by OpenStack,reviewers spent less effort and smaller amount of time to understand the code, this also, reduces the iterations. Therefore, we encourage you to use the best practices when writing code, commenting on codes, commit messages and testing."

```
An example of an acceptable function docstring is:

def mult(a: Union[intho, float],
         b: Union[int, float]) -> Union[int, float]:
```

```
    """Multiple a * b and return the result"""
    return a * b


def some_function(a: FineObject):
    """Do something with a FineObject

    :param: a is used in the context of doing something.
    """
    do_something_with(a)
```

In addition, M9 also said that OpenStack has a large code base, spanning dozens of git trees, with over a thousand developers contributing. As such, common style helps developers understand code in reviews, it also move between projects smoothly, and overall make the code more maintainable.

> M9: "One of the beauties of ESLint is that, despite there being one standardized set of rules created for OpenStack, ESLint permits the overriding of these rules on a **per-project basis** to ensure that no project is hindered by a generalised decision and projects do not have to forgo the use of eslint-config-openstack due to fears of these restrictions. Saying this however, the point of having these standardizations may be slightly defeated if the aim is not to stay as close to the common guidelines as possible."

Furthermore, M9 gave some code samples to the participants that were written and committed by core-contributors at OpenStack. He instructed the participants to explore the code, learn the style in writing code and apply the techniques they have just learned on reviewing code. Then explain their review with their mentors, and commit their reviews to the sandbox project repository.

All the participants were engaged in the activity and after the mentors have gone over their reviews, the participants posted their review on the etherpad doc.

Next, the mentors, showed couple of bad samples of commits that reviewers rejected because the commits violated the code style, which OpenStack enforces. Besides, M9 told participants to avoid such bad practices, which will certainly increase the efforts that they put in to make changes and also increase the time to

3. Insight from lead mentors

OB1 asked M1 and M2 "Can you summarize what the onboarding training is all about, and what values it brings to the OpenStack ecosystem? How has the training material evolved overtime?

M1

```
Onboarding at OpenStack is an intensive program designed for newly
graduated student in mind who are motivated and about to startn their
carrier in open source ecosystem such as OpenStack but lack the technical know-
how.
We give them materials and hands-on training that equipes them to master
the tools, which they will use in making contributions to the codebase;
add new features, fix-bugs, write documentation and participate in
working groups to OpenStack as they join a community of thousands of
developers from hundreds of companies worldwide.

Students will also learn how to use a prepared development environment
to test, prepare and upload new code snippets or documentation for review.

This year, Lenovo is sponsoring the onboarding event. But the Onboarding
is organized and run by people embedded in the community, the fast-track
course gives students a more accurate taste of what working in the
community is really like and the opportunity to ask experienced
contributors questions and gain more insight into their work with OpenStack.

There’s help beyond the classroom, too: attendees can join an ongoing
 mentoring program.

As the training evolves, our focus continually shifts towards providing a
highly interactive course where students can learn about the social norms,
modes of communication and variety of possible contributions through
 experience as opposed to lectures.
```

M2 supported M1 response and added that:

M2

```
"Onboarding new contributors is incredibly important in any open source
ecosystem and particularly one as large as OpenStack.There's a constant
flow of people joining our community and some moving on to other endeavors.

The best way to maintain a healthy community is to educate newcomers and
give them the tools they need to become effective contributors.
One of ways OpenStack does this is through the two-day long Upstream
Institute Training offered prior to each OpenStack Summit."
```

4. Project Onboarding

   Towards the end of the two-days event,

   - M1 said: "During mentoring, OUI participants are strongly encouraged (**required**) to

   join at least one project team. Once signed up to join a project team, they are assigned mentors to follow them up."

   - M2: "Project Onboarding gives participants a chance to meet some of the project team and get to know the project. Participants will learn about the project itself, the code structure/ overall architecture, etc, and places where contribution is needed. Participants will also get to know some of the core contributors and other established community members. Ideally, participants will know/ have completed the OUI basics."

5. Mentoring Program

   Prospective Participants
   All 72 participants signed up to participate for

   the mentoring program.

   P1, P2, P3, …, P72. Moreover, they indicated interest in different project teams.