

Git Collaboration

VE482 Project2 Presentation

Group 11, Yihua Liu, Yu Xiao

UM-SJTU Joint Institute

October 28, 2021

Start from a small group of 4 members

1. Owner-Contributor Mode - a centralized approach

This mode applies to the case that one main contributor does most of the work and other contributors serve as assistants of the main contributor to do minor changes, modifications, or patches.

- The main contributor also the team leader creates a repository and a `main` branch as the Owner. Usually, she or he does all of her or his work on the `main` branch and is responsible to review and merge other contributors' pull requests.
- Usually, the members directly fork the repository, make changes on their own branches, create pull requests, and wait for the main contributor to merge their changes.
- If the repository has grew very large and a patch is complex, the owner may create a `develop` branch. They will work on `develop` normally, and merge `develop` to `main` periodically.

Start from a small group of 4 members

2. Team Members Mode - a decentralized approach

This mode applies to the case that the 4 contributors are a developer team. Our Project 2 uses this method. All the members have the same rights to the repository.

- The 4 members may be divided into 2 pairs. Each pair maintains its own branches for different tasks.
- When a task is finished, one pair creates a pull request. The branch would be merged to `main` only if both pairs (all the 4 members) permit.

3. Owner-Collaborator Mode

This mode is somewhere in between. Collaborators also have the rights to merge pull requests, so they can work on their own branches and merge them without others' permissions.

When the group grows to 10

Problem

Merge conflicts may happen frequently.

Solution

- Always `git pull` first before committing to public branches.
- Use `git rebase <branch_name>` to detect merge conflicts before merging and resolve conflicts locally.
- Use more branches.

Number of branches is more related to the size/complexity of the repositories rather than number of contributors. For example, [996.ICU](#) (607 contributors) and [ohmyzsh](#) (1930 contributors) only have the `main` branch. However, to use multiple branches is always better than to use the single `main` branch.

When the group grows to 10 - Organizations

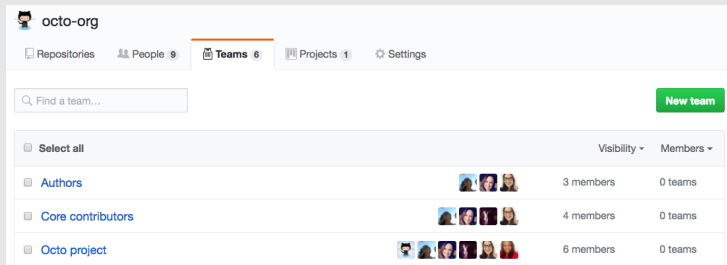
Organizations are flexible. They can be as small as 10 members, or as large as 100,000 members. For example, [Epic Games](#) has 365.4k members, and [NVIDIA GameWorks](#) has 24.6k members.

Question

Why to use organizations?

On the contrary to **user repositories** mentioned before, organizations work on **group repositories**.

When the group grows to 10 - Teams








Teams are groups of organization members that reflect your company or group's structure with cascading access permissions and mentions.

When the group grows to 10 - Teams

40 teams in the octo-org organization

Visibility ▾ Members ▾

Employees		6 members	29 teams ▴
Engineering		6 members	24 teams ▴
ApplicationEngineering		3 members	21 teams ▴
ClientSystems		2 members	3 teams ▾
Identity		2 members	11 teams ▾

octo-org octo-team





Discussions Members Teams Repositories Projects

Find a member...

Add a member

0 members 15 child team members

Role ▾

- ☐  The Octocat octocat Maintainer
- ☐  Sarah Schneider sarahs Maintainer
- ☐  Jess Hosman jhosman Maintainer
- ☐  Brett Westover bwestover Maintainer

You can even create nested teams [2].

When the group grows into 100 or even 1,000 developers

4. Owner-Collaborator-Contributor Mode

- The owner and the collaborators maintain the repository together as the core developer group. They are responsible for reviewing and merging pull requests. The core developer group will discuss any major changes by mailing lists.
- Other developers serve as contributors. They develop their own branches and create pull requests. They can discuss by IRC channels (popular servers include `freenode` and `libera`), Gitter rooms, Google Group, GitHub issue and discussion pages etc.
- More protected branches are introduced for version control.

5. Owner-Member Mode – Organization

Epic Games and NVIDIA GameWorks adapts this mode. Developers must register as a group member to view and contribute to the group repositories.

Public Project over Email

Many old, large projects accept patches via a developer mailing list [1].

```
$ git checkout -b topicA
... work ...
$ git commit
... work ...
$ git format-patch -M origin/master
0001-add-limit-to-log-function.patch
0002-increase-log-output-to-30-from-25.patch
$ cat 0001-add-limit-to-log-function.patch
From 330090432754092d704da8e76ca5c05c198e71a8 Mon Sep 17 00:00:00 2001
From: Jessica Smith <jessica@example.com>
Date: Sun, 6 Apr 2008 10:17:23 -0700
Subject: [PATCH 1/2] Add limit to log function

Limit log functionality to the first 20

---
```

Public Project over Email

Edit /.gitconfig:

```
[imap]
  folder = "[Gmail]/Drafts"
  host = imaps://imap.gmail.com
  user = user@gmail.com
  pass = YX]8g76G_2^sFbd
  port = 993
  sslverify = false
[sendemail]
  smtpencryption = tls
  smtpserver = smtp.gmail.com
  smtpuser = user@gmail.com
  smtpserverport = 587
```

Public Project over Email

```
$ cat *.patch |git imap-send
Resolving imap.gmail.com... ok
Connecting to [74.125.142.109]:993... ok
Logging in...
sending 2 messages
100% (2/2) done

(mbox) Adding cc: Jessica Smith <jessica@example.com> from
  \line 'From: Jessica Smith <jessica@example.com>'
OK. Log says:
Sendmail: /usr/sbin/sendmail -i jessica@example.com
From: Jessica Smith <jessica@example.com>
To: jessica@example.com
Subject: [PATCH 1/2] Add limit to log function
Date: Sat, 30 May 2009 13:29:15 -0700
Message-Id: <1243715356-61726-1-git-send-email-jessica@example.com>
X-Mailer: git-send-email 1.6.2.rc1.20.g8c5b.dirty
In-Reply-To: <y>
References: <y>

Result: OK
```

Large-Scale Group Work

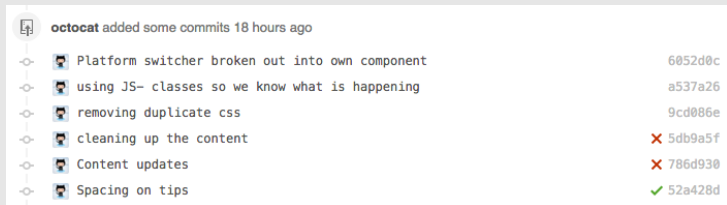
In case of unlimited contributors, the general workflow for common contributors will be:

- Fork the target repository to one's own account.
- Clone the forked repository to the local machine.
- Create a new branch for a topic and make changes.
- Commit the changes to the forked repository.
- Create a pull request.
- Make any requested changes.
- Pull request approved.

Pull Request: Status Check

Pull request is the most important part during collaboration. Normally, this part requires one reviewer to check the committed code manually. If you are using Github, then there are some useful tools that might be helpful.

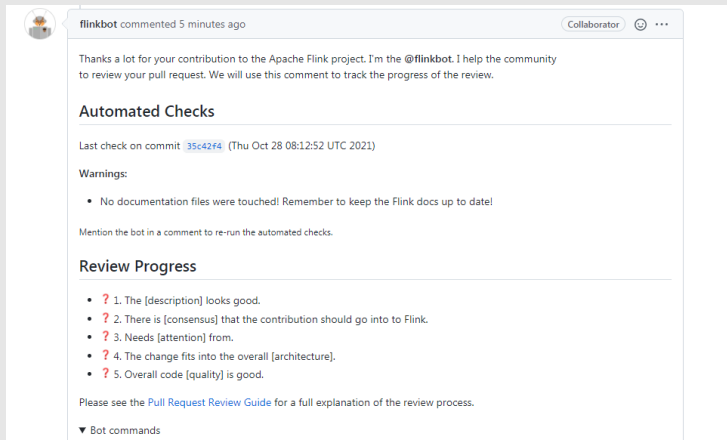
- The owner can set status checks for the repository.



- *pending, passing, or failing* state of status checks will be shown next to each commit in the pull request.

Pull Request: Github Bot

- The owner of the repository can also deploy a Github Bot (Github App) to help to do the code quality check.



A screenshot of a GitHub comment from the 'flinkbot' user, which is a bot. The comment is titled 'flinkbot commented 5 minutes ago' and is marked as a 'Collaborator' comment. The text of the comment reads: 'Thanks a lot for your contribution to the Apache Flink project. I'm the @flinkbot. I help the community to review your pull request. We will use this comment to track the progress of the review.' Below this, there is a section titled 'Automated Checks' which states 'Last check on commit 35c42f4 (Thu Oct 28 08:12:52 UTC 2021)' and lists a 'Warnings:' section with one bullet point: 'No documentation files were touched! Remember to keep the Flink docs up to date!'. It also mentions 'Mention the bot in a comment to re-run the automated checks.' Another section titled 'Review Progress' lists five items, each starting with a red question mark icon: '1. The [description] looks good.', '2. There is [consensus] that the contribution should go into to Flink.', '3. Needs [attention] from.', '4. The change fits into the overall [architecture].', and '5. Overall code [quality] is good.' At the bottom, it says 'Please see the Pull Request Review Guide for a full explanation of the review process.' and a collapsed section for 'Bot commands'.

flinkbot commented 5 minutes ago

Thanks a lot for your contribution to the Apache Flink project. I'm the @flinkbot. I help the community to review your pull request. We will use this comment to track the progress of the review.

Automated Checks

Last check on commit [35c42f4](#) (Thu Oct 28 08:12:52 UTC 2021)

Warnings:

- No documentation files were touched! Remember to keep the Flink docs up to date!

Mention the bot in a comment to re-run the automated checks.

Review Progress

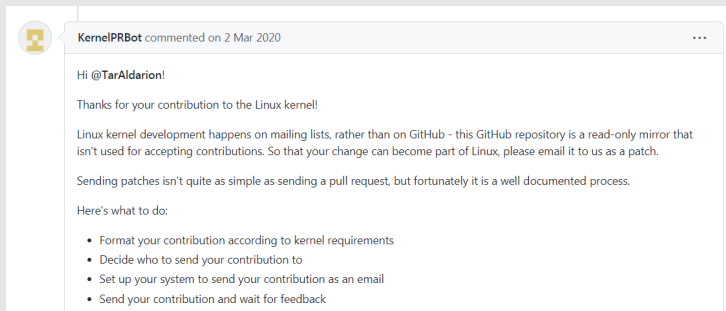
- ? 1. The [description] looks good.
- ? 2. There is [consensus] that the contribution should go into to Flink.
- ? 3. Needs [attention] from.
- ? 4. The change fits into the overall [architecture].
- ? 5. Overall code [quality] is good.

Please see the [Pull Request Review Guide](#) for a full explanation of the review process.

▼ Bot commands

Secondary Channels

Some big projects are not totally based on online development platforms like Github (e.g. linux).



Some universities also have their own git server that serves as online development platform.

Reference

- [1] Scott Chacon and Ben Straub. *Pro Git*. Version 2nd Edition. 2014. URL: <http://git-scm.com/book/en/v2>.
- [2] GitHub. *Organizations and teams*. 2021. URL: <https://docs.github.com/en/organizations>.

Thanks!