Goal of this session

- Most attendees here are highly experienced devs
- Let's discuss what we want new potential developers to know!
Brief History of MariaDB

- Created as a response to Oracle’s acquisition of MySQL

- Named after Monty’s youngest daughter Maria

- First release in Oct 2009, MariaDB 5.1

- The development is guided by the MariaDB Foundation

- Strong focus on community development, not just in-house

- Now the default MySQL variant in Debian 9
Statistics

- first pull request created 2014-06-07
- 482 pull requests created by 110 contributors
- 450 pull requests reviewed by 18 reviewers
Statistics

GitHub Pull Requests

- MariaDB 2017
- MariaDB 2016
- MySQL 2017
- MySQL 2016

© 2017 MariaDB Foundation
How to contribute?

- Similar to many open source projects
- We have our code on github.com
- We accept contributions in many forms
- Simplest are KB edits and pull requests (code)
Contributing documentation

- Go to https://mariadb.com/kb/en/
Contributing documentation

- Go to https://mariadb.com/kb/en/
- Create a new user or login
Contributing documentation

- Go to https://mariadb.com/kb/en/
- Create a new user or login
Contributing documentation

- Go to https://mariadb.com/kb/en/
- Create a new user or login
- Then open any KB article
Contributing documentation

- Go to https://mariadb.com/kb/en/
- Create a new user or login
- Then open any KB article
- Use Edit or Translate Menus on the left
- All contributions are reviewed and / or curated
Contributing code

■ Easiest way is to submit patches.

■ You will need a github account and a clone of the server.

■ The MariaDB Server codebase can be found at: https://github.com/MariaDB/server

■ Fork the MariaDB Server code on Github to your own repository: https://help.github.com/articles/fork-a-repo/

■ Create a patch and submit a pull request (we’ll do this in the workshop from scratch): https://help.github.com/articles/creating-a-pull-request-from-a-fork/
Create a patch workshop

- We are going to do a step-by-step tutorial to get a patch submitted to MariaDB server

- First create a fork of the repo.
  https://help.github.com/articles/fork-a-repo/
Step 1: Get the code

- We use a git repository. Download and install git.

  $ sudo apt install git

- Clone the server fork you have created

  $ git clone https://github.com/MariaDB/server.git

  $ cd server
Step 2: Compile the server

- Install all required build dependencies:
  
  ```bash
  $ sudo apt-get build-dep mysql-server
  $ sudo apt-get install libgnutls28-dev
  ```

- Use cmake to generate Makefiles
  
  ```bash
  $ cmake . -DCMAKE_BUILD_TYPE=Debug
  ```

- Compile
  
  ```bash
  $ make -j4
  ```
Step 3: Test the server

- We have finished building the server. We can run tests to see if it works properly.

$ cd mysql-test && ./mtr --parallel=4 --mem

- --parallel=n starts n tests in parallel
- --mem will use a memory filesystem instead of disk
Step 4: Write a patch for 10.3

- Make sure you are using 10.3 as a base:
  ```
  ~server/$ git checkout 10.3 && git pull
  ```

- Create a new branch based on newest 10.3
  ```
  ~server/$ git checkout -b 10.3-patch
  ```

- Write your patch, then add all changes and commit. Write a descriptive commit message.
  ```
  ~server/$ git add . && git commit
  ```
Step 5: Submit pull request

- Push your patch to your local github repository

$ git push

- Go to your github fork page, pull-requests tab and start a new pull request.
Step 5: Submit pull request

- Chose MariaDB/server as base fork and 10.3 branch
- Chose your fork and branch as head fork
- Click Create pull request!
Tips & tricks

- Reviews will come as comments on github.
- Address any concerns and update your branch.
- It’s strongly recommended to include test cases as part of your patch.
Creating a test case

- `mysql-test-run.pl` script will search for files ending in `*.test` in `mysql-test/t/` directory.

- It will run all `*` statements inside it as SQL queries to the server.

- It will compare output to that found in `<test-case>.result` file.

* There is a special syntax for mtr commands.
Creating a test case

- Inside mysql-test/t/ directory create a test case file.

```
~/server/$ cd mysql-test && touch t/hello.test
```

- Add statement within test file

```
$ echo 'SELECT "Hello World!";' > t/hello.test
```

- Run mysql-test-run on the new test.

```
$ ./mtr hello
```
vicentiu@vicentiu-desktop:~/Workspace/mysqld-test$ ./mtr hello
Logging: ./mtr hello
vardir: /home/vicentiu/Workspace/MariaDB/mysqld-test/var
Checking leftover processes...
Removing old var directory...
Creating var directory '/home/vicentiu/Workspace/MariaDB/mysqld-test/var'...
Checking supported features...
MariaDB Version 10.2.10- MariaDB-debug
- SSL connections supported
- binaries are debug compiled
Collecting tests...
Installing system database...

<table>
<thead>
<tr>
<th>TEST</th>
<th>RESULT</th>
<th>TIME (ms) or COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>worker[1] Using MTR_BUILD_THREAD 300, with reserved ports 16000..16019 SELECT &quot;Hello World!&quot;;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hello World!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hello World!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>main.hello</td>
<td>[ pass ]</td>
<td></td>
</tr>
</tbody>
</table>

The servers were restarted 0 times
Spent 0.000 of 7 seconds executing testcases
Completed: All 1 tests were successful.

vicentiu@vicentiu-desktop:~/Workspace/MariaDB/mysqld-test$
Creating a test case

- If the results of the test are ok, you need to record it.

```
$ ./mtr --record hello
```

- This will create the file `mysql-test/r/hello.result`

- When submitting a patch, include both `.test` and `.result` files.
Make your own plugin workshop!

- We’ll start from a sample plugin and extend it.
- We’ll create test cases for it.
- Prepare the contribution to be submitted as a pull request.
- Submit a pull request to the main server.
Buildbot

- MariaDB is built continuously using our buildbot infrastructure.


- All main branches (5.5, 10.0, 10.1, 10.2, 10.3) are built fully on many builders.

- All dev branches starting with bb-* as a name are built on a subset of builders.
Jira & IRC & Community!

- We work in 1 to 2 week sprints.
- Use Jira [https://jira.mariadb.org](https://jira.mariadb.org) to track Bugs, Pull Requests and New Feature requests
- You can contribute by filing bugs too!
- Find us on IRC #maria and #maria-dev on FreeNode
- Mailing Lists:
  - Maria Developers [https://lists.launchpad.net/maria-developers/](https://lists.launchpad.net/maria-developers/)
  - Maria Discuss [https://lists.launchpad.net/maria-discuss/](https://lists.launchpad.net/maria-discuss/)
License

- MariaDB Contributor Agreement
  https://mariadb.org/get-involved/getting-started-for-developers/mca/
  https://mariadb.org/get-involved/getting-started-for-developers/mca-faq/

- BSD-new
  https://en.wikipedia.org/wiki/bsd_licenses
Frequent Problems

- git push failed with permission denied
  - remote is set to MariaDB/server instead of <own-repo>/server

- When trying to start mysqld, error messages file not found.
  - pass --lc-messages-dir=<path-to-sql/share/>

- After installing missing dependencies, configure fails with same message
  - Remove CMakeCache.txt, rerun cmake
Improve the experience for new contributors

- We are missing a server architecture doc
  - We can point to MySQL ones, but it's best if we'd create design documents for our major features

- Beginner-friendly bugs?
  - There is a jira label for that. Hard to find, ideally embed it on mariadb.org

- Fix coding style errors across the whole code base once and for all?
Script to start mysqld from source folder

- start_mariadb_instance.py

usage: start_mariadb_instance [-h] [--force] [--extra EXTRA]
                        mysqld_binary sourcedir datadir

positional arguments:
  mysqld_binary    path to compiled mysqld binary
  sourcedir       path to base source repository directory
  datadir         path to where the data directory for the server will be

optional arguments:
  -h, --help       show this help message and exit
  --force          install database, even if datadir already exists
  --extra EXTRA    extra arguments to pass to server, verbatim

- Will print PID of mysqld and it's output
Thank You!

Contact me at:

vicentiu@mariadb.org
vicentiu@ciorbaru.io

Blogs:

mariadb.org/blog
vicentiu.ciorbaru.io
How do I do X?

Problem: Multiple ways of doing X

Everybody has their own way, hard for somebody new to learn.
How to start making sense of everything?

- Create a simplified mental model of the system.
- Use the model to make sense of each git command.
- Adjust model when it's not complex enough.
The git commit

- Everything in git revolves around the "commit".
- What is a commit?
The git commit

Commit Hash: 613dd62a76b51df38b96e36bce5bd8b7be2ca73a
Commit Author: Sergey Vojtovich
Commit Date: Thu Aug 10 15:45:03 2017 +0400
Commit Message:

MDEV-1153 - Introduce status variables for table cache monitoring and tuning

Status variables added: Table_open_cache_hits, Table_open_cache_misses, Table_open_cache_overflows, Table_open_cache_active_instances.
The git commit

diff --git a/mysql-test/r/status.result b/mysql-test/r/status.result
index 9a8a5bd..9b82c78 100644
--- a/mysql-test/r/status.result
+++ b/mysql-test/r/status.result
@@ -386,6 +386,29 @@ Handler_tmp_write 2
    Handler_update 0
    Handler_write 0
    drop table t1;
+##
+## MDEV-11153 - Introduce status variables for table cache monitoring and tuning
+##
+##

Commit Diff
The git commit

■ Every git commit has **at least** one parent.

■ The commit hash is computed using all the info from last slide **and** it's parent(s).

■ There is only one commit which doesn't have a parent, the initial commit from the repository.

■ All git commands revolve around this definition.

■ We can obtain any version of the code using this definition.
The git branch

- A branch is a pointer to a commit.
- Whenever a new commit gets created, the current working branch gets updated to the new commit id.
Rule #1: Basic Commit Rules

- Every commit should compile.
- Every commit should have all tests pass.
- Every commit should be a self-contained, logically valid change.
- In short: "DO NOT BREAK THE BUILD!"
Rule #2: Commit content

- Commit often, small changes. (Respect Rule #1)

- Make commit messages relevant.
  - One should be able to tell roughly what a commit does without reading the code.
  - One should be able to understand the reasoning for a commit from its message.
Rule #2: Commit content

- Commit messages must follow the pattern:
  - Commit title (max 80ish chars)
  - Empty line
  - Commit message (max 80 chars)

- When making a commit that fixes a particular Jira Bug, the format should be:
  MDEV-XXXXX <MDEV-XXXXX-Title>

- Tools that work with git expect this format. (Jira)
- Makes life easier for everybody.
Rule #3: Branches are cheap, use them

- A branch is just a pointer to a commit.
- Help organize different bugfixes / features.
- When working on a bugfix:
  - Fetch the current tip of the main branch.
  - Make another branch based on that.
- Avoid working on the main branch directly.
Rule #3: Branches are cheap, use them

$ git checkout 10.3

$ git pull 10.3  # Now we are up to date

$ git checkout -b bb-10.3-MDEV-XXXXX

# Now we are working on a separate branch.
Rule #4: Prefer Rebasing Over Merging

- When working on a bugfix for a certain version, somebody may push something in the meantime.

- When trying to merge to the main branch, you will need to update.

- If you don't rebase, an extra merge commit will be generated.
Rule #4: Prefer Rebasing Over Merging

- Merge commits are usually not desireable
  - Integrate a large set of changes directly.
  - Difficult to pinpoint which change causes a bug.
  - Merges contains all conflict resolution information. => Impossible to read diffs.
Rule #4: Prefer Rebasing Over Merging

$ git checkout 10.3

$ git pull 10.3  # Now we are up to date

$ git checkout -b bb-10.3-MDEV-XXXXX

$ git rebase 10.3

$ git checkout 10.3

$ git merge bb-10.3-MDEV-XXXXX
# This will just update 10.3 branch pointer
# to be the same as bb-10.3-MDEV-XXXXX
Rule #5: Rebase interactive to clean-up

- When your commit history is "dirty"

- Rewrite it with `git rebase --interactive`
  - Reorder commits
  - Change commit messages
  - Merge multiple commits into one
  - Drop unneeded commits

# Chose a commit before your patches

$ git rebase -i HEAD~10

- Try to not break Rule #1 && Rule #2
  - Commits must be self contained & not break tests.