Code review is a critical part of our development process. We are making the following adjustments to allow us to continue to maintain and improve our code quality / integrity. Purpose for this guideline is to ensure the process is streamlined.

General principles:

1. Create a branch in your fork for each change you make, and make sure it is rebased onto the latest **master** when you open the pull request
    a. If you need some best practices here feel free to ask (Josh, Tim, Hao)
2. Avoid large pull requests covering more than one features or fixes. Do it incrementally.
3. Close requests and resubmit if they are left unattended for more than 3 work days.
   unattended

Preparing For Review:

1. Read and follow our Code Style Guide.
2. Write easy-to-review code. Readability of your code should be a priority while writing all code. Adding file, class, and inline comments will help both reviewers as well as future code maintainers.
3. Focused, single-function PRs. Avoid fixing multiple bugs or adding multiple features within a single PR. Focused PRs are easier to test and verify.
4. Review your own code before sending PR. A quick reading of your own code can often find obvious typos or potential bugs. When reviewing your own code, keep a mindset of someone who is seeing the code for the first time. This can often bring out confusing or overly complex code which can be improved before sending the PR.
5. Think about how CI can test more of your PR (so that people don't have to). If we can get cars/trucks to self-drive we surely can make our CI self-driven :)

Prepare PR for Submission:

1. Choose a descriptive title of the pull request.
2. In the PR description, make sure you summarize the purpose of the PR, changes in interface / behavior, key logic / algorithms implemented with references (if not already included in code comments), and any key aspects of the code change that you would like the reviewers to focus on. Remember, the value of the review is to solicit feedback, not to pass the review. It is strongly recommended to include the related git issue #s in the description.
3. Make sure you consolidate the commits into one.
4. Assign appropriate labels to the request. They should reflect the components affected and the project milestone. Label it with P0 if it requires immediate attention from reviewers.
5. Select the appropriate reviewers. See the table below or Git CODEOWNERS file for the default reviewers. If you don't know who should review, ask the project owner or your peers. The general guideline is to include those who have the best knowledge of the code touched and those who will be directly affected by the changes made.
6. Reference the related Issues in both the git commit message and the PR description. See Autolinked references and URLs.
7. Do not rely on PR for communication. Reach out to the reviewers directly if you would like it to be expedited.

During Review (Submitter):

1. Watch review feedback comments and respond accordingly. Even if you don't agree or chose to make no code change based on feedback, respond to reviewer comments.
2. After making any code changes, notify the reviewers by commenting. Adding "PTAL" is often used to ask reviewers to Please Take Another Look.
3. Code reviews are an open discussion. Do not take all feedback as an order to follow. Disagreement is part of the process, but you should explain why you disagree and allow your reviewer to respond. Unresolved disagreements may be best solved offline.
4. Do NOT self merge.

During Review (Reviewers):

1. Be timely. Review P0 requests ASAP. Review P1 requests within 24 hours. Review all other requests within 2 days. If you will be unable to review within the expected time, add comment to the PR explaining such.
2. Correctness. Confirm the code fixes the bug or adds the desired feature. Follow the logic and try to predict functionality and ensure the result is desired. Think of edge cases or potential unexpected input. This is an opportunity to find bugs before they cause problems.
3. Big Picture. All new code must fit with existing design in a logical way. The interaction between main components of the code should be obvious and correct. While a flexible architecture is desired, be careful to push back on over-engineered components.
4. Review test coverage. Ideally, all code should have appropriate tests (unit and/or component level). Test code should be reviewed with the same focus as functional code. Tests need to be readable and maintainable. Ensure tests are abstracted properly, avoiding unnecessary dependencies.
5. Style. Finding style problems is important to keep the code base consistent, readable, and bug-free. Comment on style problems with a link to the specific section of the Style Guide. Also remember the Style Guide is a living document, so if an update to the Style Guide would help, please submit a request to update the Style Guide.

Post Review (Reviewers):

1. Complete final review of the PR when there are at least one +2 or two +1 approving the request and no pending change requests.
2. Merge the PR.

Recommended Reviewers:

The owners are chosen based on their familiarity with the current code base. We will expand the list as more people are involved in leading the development work of each component.

Repo/Branch	Component	Recommended reviewers (+2)
default		Hao, Tim, Amit, Pingliang
drive/common		Eugene, Wenbin
drive/master	., cmake, docker	Josh, Jia, Wenbin
	common	Tim
	control	Amit
	map, plusmap	Wei, Mianwei
	perception	Anurag, Di, Wenbin
	prediction	Di
	planning	Amit, Fan
	runtime, tools	Amit, Owen, Jinwen, Pingliang
	ui	Pingliang
simulation/master		Pingliang
models/master, model/master		Mianwei, Di, Anurag
external/master		Josh, Wenbin
docker/master		Josh, Wenbin
github-webhook-proxy/master,		Josh, Jia

github-sync		
basedev/master		Josh, Jia, Wenbin
jenkins-jobs/master		Josh, Jia, Wenbin
DRIVEOS/master		Wenbin, Wenjun
hdmap-generation/master		Wei
ansible/master		Josh, Mingming
tools/master		All +1's

Links

https://wiki.openstack.org/wiki/GitCommitMessages