# SIG CLI Issue Backlog

Grooming work for new and existing contributors

- Link: https://goo.gl/YEq33R
- Author: @pwittrock
- Last updated: 10/25/17

## Background

A goal of SIG CLI is to keep the SIG open to contributions from anyone willing to do the work to get involved. While many folks have shown interest in contributing, we have struggled to keep most folks engaged. Kubernetes recently conducted a survey of new contributors, and major themes were that:

- Contributors don't know where to start
- There are too many details to learn before contributing
- Communication is hard / everyone is too busy to help
- Hard to get reviewers on PRs

These challenges can be reduced by:

- Providing a backlog for contributors to browse and pull work from.
- Scoping work that can be done with minimal experience so folks can pick up and work on it.
- Marking issues as good first time issues if they can be done by someone with no experience.
- Reducing the need for constant communication by having the work be well defined and clearly scoped in the issues themselves.
- Ensuring each issue has a stake holder that is committed to seeing that changes are reviewed.

## Contribution lifecycle

1. A good issue is created with description and labels.
2. SIG agrees that work for issue will be accepted.
3. Issue moved to the *backlog* column in the New Contributors Project.
4. Contributor assigns issue to self, or asks issue be assigned if they are not a Kubernetes org member.
5. Issue moved to the *assigned* column in the New Contributors Project.
6. Contributor updates issue weekly with status updates, and pushes work to fork
    - Periodic feedback provided
    - Discussion between contributor and stakeholder occurs on issue
7. Contributor sends PR for review
    - Stakeholder ensures the appropriate reviewers exist
    - Discussion and updates occur on the PR
8. PR accepted and merged

## What makes a good issue?

### Stakeholder and Contributor

A stakeholder typically files the issue, wants to see the work done, and will find reviewers for PRs that address the issue.

The contributor is the issue assignee - they provide PRs for review to close the issue.

The stakeholder may become the contributor. They must find a new stakeholder to review the work and help follow through on issue closure.

### Encapsulated

Issues that require modifying large pieces of existing code are typically hard to accept without multiple reviewers, require a high degree of communication and require knowledge of the existing codebase.

This makes them bad candidates for contributors looking to get started independently.

Issues with good encapsulation have the following properties:

- Minimal wiring or changes to existing code
- Can disable / enable with a flag
- Easy to review the contribution on its own without needing to examine other parts of the system
- Low chance of needing to rebase or conflicting with changes made in parallel

### Consensus on work within the SIG

Work described in issues in the backlog should be agreed upon by the SIG. PRs sent for review should have the code reviewed, not the *reason* for doing the PR.

SIG CLI needs to come up with low overhead a process for accepting proposed work.

1. Create an issue for the work
2. SIG agrees to accept work for the issue (as described) if it is completed
3. Add issue to the issue backlog

## Types of code contributions

### Code documentation

Documenting code is an excellent starter task. It is easier to merge and get consensus on than writing tests or features. It also provides a structured approach to learning about the system and its components. For the packages that need it most, understanding the code base well enough to document it may be an involved task.

- Adding doc.go files to packages that are missing them
- Updating doc.go files that are empty placeholders
- Adding examples of how to use types and functions to packages
- Documenting functions with their purpose and usage

### Test coverage

Improving test coverage and augmenting e2e tests with integration tests is also a good candidate for 1st and 2nd time contributors. Writing tests for libraries requires understanding how they behave and are meant to be used. Improving code coverage allows the project to move more quickly by reducing regressions issues that the SIG must field, and by providing a safety net for code reviewers ensuring changes don't break existing functionality.

- Write unit tests for functionality currently only covered by integration and e2e tests. > Integration tests may run processes, such as the apiserver, but do so locally. > E2e tests run a full Kubernetes cluster (remote).
- Write integration tests for functionality currently only covered by e2e tests.
- Improve coverage for edge cases and different inputs to functions.
- Improve handling of invalid arguments.
- Refactoring existing tests to pull out common code into reusable functions. > *This should be very well scoped as it impacts existing tests and reviewers need to make sure nothing breaks.*

### New libraries

Encapsulated libraries (collections of functions devoted to one simple purpose - e.g. date/time utils) are great contributions for experienced contributors - either programming in Go, or with Kubernetes.

Because the libraries are encapsulated, it is easier for reviewers to determine the correctness of their interactions with the existing system. If the functionality is new or can be disabled with a flag, the risk of accepting the change is reduced, improving the chance the change will be accepted.

### Modifying existing libraries

Tasks to perform non-trivial changes to existing libraries should be reserved only for folks who have made multiple successful contributions of code - either tests or libraries. PRs to modify existing libraries typically have multiple reviewers, and can have subtle side effects that need to be carefully checked for.

Improvements in documentation and testing (above) reduces the burden to modify existing code.

## Managing a backlog issue

### Setting Labels

For contributors to pick up new tasks independently, the scope and complexity of the task must be well documented on the issue. We use labels to define the most important metadata about the issues in the backlog.

#### Size

- size/S > 4-10 hours
- size/M > 10-20 hours
- size/L > 20+ hours

#### Type (docs / tests / feature)

- type/code-cleanup > Usually some refactoring or small rewrites of code.
- type/code-documentation > Write doc.go with package overview and examples or document types and functions.
- type/code-feature > Usually a new go package / library for some functionality. Should be encapsulated.
- type/code-test-coverage > Audit tests for a package. Run coverage tools and also manually look at what functions are missing unit or > integration tests. Write tests for these functions.

#### Description

- Clear description of what the outcome is
- Pointers to examples if they exist
- Clear stakeholder who will be responsive to the issue and is committed to getting the functionality added

## Assigning an issue once work has started

1. Contributor messages stakeholder on the issue, and maybe on slack
2. Stakeholder moves issue from backlog to assigned
3. Contributor updates issue weekly and publishes work in progress to a fork > The issue should be updated with a link to the fork.
4. Once work is ready for review, contributor files a PR and notifies the stakeholder

## What is expected as part of contributing a library?

### Documentation

- doc.go
- Comments on all functions and types

**Tests**

- Unit tests for functions and types
- Integration tests with a local control plane (forthcoming)
- Maybe e2e tests (only a couple)

**Ownership of addressing issues**

- Fix bugs discovered in the contribution after it has been accepted