Release Management for LFN Projects

THE LINUX FOUNDATION

LF NETWORKING

Contents

- Part I: Introduction
  - What is Release Management?
  - Audience
  - Roles and Responsibilities
- Part II: Developing a Release Process
  - Identify the work products
  - Document the process steps
  - Identify logical checkpoints
  - Document the tasks
  - Prepare a schedule template
  - Socialize the proposed release process
  - Seek TSC approval
- Part III: Executing the Release Process
  - Preparing a release schedule
  - Sub-Project release plans
  - Tracking release progress
  - Milestone status reviews
  - Managing exceptions
  - Post-release retrospective
- Part IV: Next Steps

Part I: Introduction

What is Release Management?

THE LINUX FOUNDATION

LF NETWORKING

What is release management?

According to Wikipedia:

"Release management is the process of managing, planning, scheduling and controlling a software build through different stages and environments; it includes testing and deploying software releases."

This is a fairly broad definition. So, more specifically, what does release management mean in the context of a Linux Foundation Networking (LFN) project?

What is release management?

For the purposes of this training, we are going to tighten the Wikipedia definition to say that we are referring to:

- Collections of projects
- Having subproject interdependency
- Simultaneously released
- On a periodic basis

Let's look at each of these characteristics, individually.

Collections of projects

Many LFN "projects" are, in fact, "projects of projects".

This can be confusing, because we often use the term "project" to refer to both the parent project (e.g., ONAP) as well as to sub-projects (e.g., CCSDK in ONAP).

For our purposes, we are referring to release management at the parent project level.

Collections of projects

It may have already occurred to you that individual projects do their own release management.

This is very true and is the key to understanding the difference between "traditional" release management with which you may already have had experience, and release management for a LFN project of projects.

You can think of this type of release management as a "release of releases" or a "meta release".

Project interdependency

For this training, we are also going to focus on projects where the sub-projects have interdependency. This is largely what will drive our release management planning.

Examples:

- Multiple projects form a pipeline that supports a process flow
- One project produces an API used by other projects
- One project produces a test framework used by other projects to validate their software
- One project maintains a documentation framework used by other projects

Simultaneous release

By “simultaneous release” we mean that the sub-projects coordinate to produce a meta release at the same time according to an approved release plan and an approved schedule at the parent project level.

Periodic release

- Regular, predictable releases are the lifeblood of open source projects. It's difficult to maintain interest in a project whose releases are infrequent and/or irregular.
- Irregular releases can also signal to end users that the project is having problems, is not well run, and may not be sustainable.
- A common cadence for projects in LFN is 6 months, or twice per year.
- However, the specific cadence isn't as important as it is to have a regular cadence and to be predictable.

Audience

Audience

This training is intended for the following:

- A need for a simultaneous release that is supported by the community
- A willingness by the community to commit to a meta release with a consistent cadence
- Projects lacking a reliable or satisfactory release plan
- A willingness by at least two community members to volunteer as community release managers

Audience

This training may not be necessary or suitable for:

- Projects that already have an approved release plan that is satisfactory to the community and that has been used to successfully complete several releases, following a regular, predictable cadence
- Projects that lack support from the community for simultaneous releases, or releases that follow a consistent cadence
- Projects that lack any members willing to serve as community release managers

Release Management Roles and Responsibilities

THE LINUX FOUNDATION LF NETWORKING

Linux Foundation Staff

As part of a community self-sufficiency effort, LFN staff are moving away from providing direct, dedicated release management support. This is more consistent with how most LF communities function.

A LFN staff member will do the following:

- Provide this training to the community
- Assist community release managers in developing a release process
- Assist community release managers in developing release schedules
- Provide input on specific issues that arise
- Participate in release retrospectives and help identify and recommend improvements to the release process

Community Release Managers

We recommend at least two community release managers. This helps reduce the workload and provides for ongoing support in the event that one community release manager is unavailable.

Community release managers will do the following:

- Develop and maintain the release process
- Develop and maintain the release schedule and other release documentation
- Track the completion of release management tasks
- Periodically present the release status to the TSC and the community and make recommendations about approving release milestones
- Help the TSC to resolve issues related to incomplete release management tasks, missed milestones, and other exceptions
- Make a recommendation to the TSC on release Sign Off
- Lead the community in a release retrospective following release Sign Off and recommend changes to the release process

Project Technical Lead (PTL)

The PTL will do the following (or delegate):

- Update the milestone status tracker as release management tasks are completed
- Alert the release manager and the TSC to issues, disruptions, or risks to the release schedule, or to particular requirements planned for the release
- Complete a sub-project release plan for each release
- Complete exception requests, as necessary
- Participate in the release retrospective

Technical Steering Committee (TSC)

Note: the TSC may do this directly, or it may delegate to another person or group.

- Review and approve the release process, as well as significant changes
- Review and approve the release schedule, as well as changes
- Review and approve milestone completion recommendations from the release manager
- Review and approve the signoff recommendation from the release manager
- Approve milestone exceptions
- Contribute to the release retrospective

Adapting the Training to the Community

THE LINUX FOUNDATION

LF NETWORKING

Adapting the Training to the Community

As with most things in life, there's more than one valid way to do release management.

The following sections will cover the development and execution of a release management process. The material in these sections is based on what has been successful in other LFN communities over the years. However, that does not mean that it's the only way to do it.

The material in this training is meant to get you thinking in depth about release management and to suggest ways to do it. However, it is not "required".

Ultimately, each community will need to decide how to do release management.

You may decide to do everything in this training exactly as described, or you may want to pick and choose which processes to adopt. You may want to adopt some processes, but with changes that make sense for your community.

Adapting the Training to the Community

However it's done, in the end, we need to make sure that the release management process adheres to the following principles:

- Is transparent and informative - The community should be able to understand the content and the status of the release, throughout the release cycle, even without the release manager
- Is independent of any individual or group - The status of the release should be independently verifiable by any knowledgeable community member, using readily available data, and should not depend on any individual or group.
- Is feasible, efficient, and not burdensome - The release management tasks should be straightforward and should strike a balance between providing transparency and not burdening the RMs or the development teams too much.

Part II: Developing a Release Process

What is a Release Process?

In Part I, we talked about the definition of release management. In Part II, we're going to dig into the release process. But what exactly is a "release process"?

For our purposes, we are going to say that we have a well defined release process when we have the following:

- A list of work products that will be produced as a result of successfully completing the release process
- A documented workflow that shows the steps and dependencies for each work product
- A set of milestones with allocated workflow steps
- A set of tasks to meet the requirements for each milestone
- A release cadence and a schedule template with the milestones
- TSC approval

Release Process Development Overview

We will develop the release process as follows:

1. Identify the work products
2. Document the process steps and dependencies for each work product
3. Identify logical checkpoints that may be used as release milestones
4. Document the tasks that must be completed for each milestone
5. Prepare a schedule template with the identified milestones
6. Socialize the proposed release process, get feedback, and iterate
7. Seek TSC approval

Identify Work Products

THE LINUX FOUNDATION

LF NETWORKING

Identify Work Products

- This might seem obvious, but identifying the work products that are expected from a release process is a critical first step
- It may be that you only intend for some portion of your project's work products to be produced by a particular release process
  - Some work products may need a different process/cadence or are what we refer to as "unmanaged"
- Also, our work products, and thus our release management tasks evolve over time
  - Sub-projects leave or join the parent project
  - Existing sub-projects add, remove, or change work products

Identify Work Products

- Since our release management tasks change when our work products change, it's important to identify and update the work products that will be produced by a given release cycle
- One way to keep track of this is for sub-projects to assert their participation at the start of a release cycle and to provide a sub-project release plan that documents their work products.
  - We will discuss this more in Part III: Release Process Execution and Administration

Identify Work Products

We've discussed "work products" generically, but what exactly is a work product? A work product includes a description of the content and format.

For example:

- Software component delivered as a docker image
- Software component delivered as an RPM
- Configuration file delivered as yaml
- Computer code delivered as a link to a GitLab repository
- Documentation delivered as read-the-docs
- Specification delivered as a PDF

Note that it's up to the TSC to establish guidelines for acceptable delivery methods and formats.

Document Work Process Steps and Dependencies

THE LINUX FOUNDATION LF NETWORKING

Document the Work Process Steps and Dependencies

- Now that we have our work products identified, we need to document the steps to create that work product, as well as any dependencies.
- Typical steps for a software component might include:
  - Requirements definition, documentation, approval
  - Software development including new features and technical debt
  - Component test
  - Integration test
  - Documentation
  - Package (docker image, RPM, etc.)
- If your project has requirements, such as TSC or subcommittee reviews or approvals, license scans, code scans, coverage analysis, etc. be sure to include these, as well

Document the Work Process Steps and Dependencies

- We also need to document dependencies, especially those on external artifacts, or between work products internal to the project.
- For example:
  - Integration testing of components depends on updates to the test framework
  - Downstream components of a pipeline must wait until upstream component development is complete in order to complete their development
  - A component has a dependency on a release artifact from an upstream (external) project

Document the Work Process Steps and Dependencies

- Don't worry about getting the steps and dependencies perfectly documented.
- Release management is an iterative, continuous improvement process.
- We will create a process with our best effort, execute it, take careful notes, then plan updates for the next release cycle.
- With diligence, over time, our process will become better, and it will also adapt to changing requirements and conditions

Identify Milestones and Allocate Tasks

Identify Milestones and Allocate Tasks to Milestones

- Now that we've identified the tasks required to complete each work product, we need to create milestones.
- Milestones are check steps along the way through the release process that give us an opportunity to check the health of the release and to provide transparency to the TSC and the community on the status of the release.
  - We will talk about evaluating release status at milestones in Part III
- We don't want to have separate milestones for each work product, so we are looking for commonality across work products.

Identify Milestones and Allocate Tasks to Milestones

A note about the number of milestones to use:

When developing a release process, it is easy to add lots of milestones, especially if you haven't had experience executing a release process before. However, it is important to remember that every milestone comes with overhead. The release manager must evaluate the status and make a recommendation to the TSC for each milestone. The TSC must plan time to evaluate the recommendation and approve the milestone.

Therefore, you must balance the need for transparency and timely status reporting with the time limitations of the release manager and the TSC. After all, the TSC has a lot to do and can't afford to spend all of its time on release management.

Typically, about 5 or 6 milestones (not counting Kickoff - M0) seem to be a good number for most projects. However, this is ultimately the decision of the release manager and the TSC.

Identify Milestones and Allocate Tasks to Milestones

It's up to the community to determine what milestones to use. While there may be milestones that are unique to a project, some common milestones are:

- Requirements definition
- Feature complete
- Component test
- Integration test
- Documentation
- Packaging

Note: it's not necessary to have a milestone name or definition. Sometimes a milestone is simply a convenient checkpoint for a group of dissimilar activities.

Identify Milestones and Allocate Tasks to Milestones

Remember that we want our milestones to be:

a) Common across projects (as much as possible); and

b) Verifiable. Sometimes, a milestone looks good on paper, but is very difficult to verify in practice. These should be avoided.

Identify Milestones and Allocate Tasks to Milestones

Next, we take the work process steps that we identified earlier and break them into more detailed tasks, then assign these tasks to our milestones.

Note about documentation tasks:

We all know that documentation is often tacked onto the end of a development cycle. This can create multiple problems and release delays. One way to avoid this is to create documentation tasks throughout the release process so that there is less to do by the end of the release. For example, you could have a "preliminary documentation" task near the middle of the release cycle, then a "final documentation" task at the end. Or, if your documentation process can be easily broken down into common individual steps, then you could assign those steps as tasks across the release cycle. However you do it, it's wise to get the team to do some documentation work earlier in the release cycle so that it doesn't all have to be completed at the end.

Identify Milestones and Allocate Tasks to Milestones

- As we create tasks and allocate them to milestones, keep in mind the following:
  - Is the task necessary, or is it just nice to have?
    - Community members are busy and are often working on the project part time.
    - Each task is additional overhead to the team members and the release manager.
    - Make sure that the task is worth the effort.
  - Is the task feasible with a reasonable amount of effort?
    - Similar to milestones, tasks can look good on paper, but be difficult to execute in practice.
    - Does the team have the expertise and the tools to complete the task?
      - If not, reconsider whether the task is necessary. If it is, develop a plan to acquire the necessary resources.
  - Have we described the task with sufficient detail?

Identify Milestones and Allocate Tasks to Milestones

As we develop tasks, we need to document them to ensure that we have a common understanding of what the task is. This will help when we socialize our release plan with the community and seek approval from the TSC.

A good way to do this is with a wiki table. For example:

Task ID	Milestone(s)	Summary	Detailed Description	Required Resources

Prepare a Schedule Template

THE LINUX FOUNDATION

LF NETWORKING

Prepare a schedule template

Now that we've established our milestones and tasks, the next step is to rough out a schedule template.

By "template", I mean that we don't attach specific dates to the milestones, we just use time offsets, like this:

Milestone	Offset
M0	0
M1	M0 + 4 weeks
M2	M1 + 7 weeks
etc.	

Prepare a schedule template

You may need to consult with team members who have had experience building the work products in order to get an accurate idea of the task durations and the appropriate milestone offsets.

Don't worry about making it perfect. We will collect information during the first release cycle with the new release plan that will help us make adjustments over time.

Once we are satisfied with the template, we can start to get an idea of what the release cadence should be. For example if the template shows a total time of 20 weeks, then a 6-month release cadence is probably appropriate.

Remember that we will have down time for holidays and community events, so if the time is exactly 26 weeks, then we're unlikely to finish during a 6 month period.

Socialize the Release Process and Seek TSC Approval

THE LINUX FOUNDATION

LF NETWORKING

Socialize the Release Process

We now have all of the elements of a release process that we discussed at the beginning of Part II. Take a moment to review the list again.

Now we need to socialize the documented release process with the community and solicit feedback.

Note that it is not necessary to wait until every element is ready to begin sharing the proposed release process. For example, you might start sharing the task definitions before you've finished the schedule template.

Before approaching the TSC, you may want to attend sub-project or sub-committee meetings and present the proposed process. You may want to reach out to key individual contributors and ask for their feedback.

Another approach is to set up a regular weekly community meeting as you're developing the process so that you get feedback in real time.

As you receive feedback, iterate the release process, and share the update. This may take some time, but it's important to get input from most, if not all, of the stakeholders.

Seek Approval from the TSC

Once you've socialized the release process and received feedback from most of the stakeholders, it's time to present the proposed process to the TSC.

Since you've already put in a lot of time sharing and getting feedback, it's likely that many of the TSC members are already familiar with the proposed process. At this point, the approval should be just a formality.

Request time on the TSC agenda, and on the scheduled date, make the presentation. Walk through each of the elements of a release process that we discussed in this Part.

In addition to describing the proposed process, be sure to summarize your efforts to socialize the process and get feedback. TSC members will feel more confident in approving the process if they know that it has been reviewed by the community, including key individual contributors.

Once the TSC has approved the process, make sure that the approval is documented in the meeting minutes.

Frequently Asked Questions (FAQ)

Q: This seems like a waterfall style of planning, we want to use agile planning.

A: Remember that this is a meta release. So, we're talking about a release of releases. The individual sub-projects are free to do their releases as they choose, just as long as they provide deliverables into the meta release per the schedule.

Q: We want to do more frequent releases than the project cadence.

A: Again, this is a release of releases. Individual sub-projects are free to follow a more frequent cadence if they choose, as long as they provide deliverables into the meta release per the schedule.

Part III: Executing the Release Process

Introduction

Now that we have our release process, it's time to apply it.

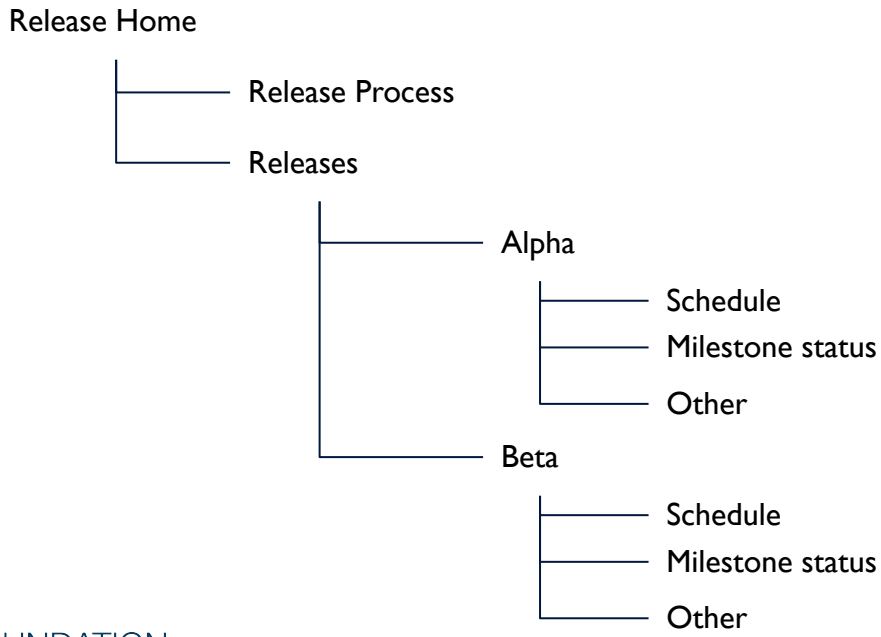
We will examine the following aspects of executing a release process:

1. Preparing a release schedule
2. Sub-Project release plans
3. Tracking release progress
4. Milestone status reviews
5. Managing exceptions
6. Post-release retrospective

Introduction

Throughout Part III, we will assume that release information is being documented on a wiki.

A common way to organize release information on a wiki is as follows:



Preparing a Release Schedule

THE LINUX FOUNDATION

LF NETWORKING

Preparing a Release Schedule

As discussed in Part II, one of the elements of the release process is a schedule template.

For the first pass at a schedule, we will simply pick a TSC meeting date as our starting point (aka "Kickoff", M0), then start assigning dates to the rest of our milestones, using the offsets from our template to determine the specific date for each milestone.

A good practice to follow is to schedule each milestone so that it falls on the same day of the week as the TSC meeting. That simplifies the TSC review of the milestone status.

So, for our next pass at the schedule, we are going to adjust the milestone dates so that they fall on the nearest TSC meeting date. If the offsets are in whole weeks, then we may already have this condition, since we set M0 to a TSC meeting date.

Preparing a Release Schedule

For our final pass at the schedule, we need some data. We need a list of major international holidays that affect our community. LF projects are international, but often have people from some regions more than others, so we just need to focus on the regions where most of our community resides.

In addition to holidays, we also want to create a list of major technical events that our community attends. Many of these will be LF events, but some will likely be events from other organizations, as well.

Now add these events to the schedule in groups. This is where a Gantt chart comes in handy. It makes it easy to see where there are conflicts between milestone dates and holidays or technical event (example).

Preparing a Release Schedule

Finally, adjust the milestone dates so that they don't occur during, immediately before, or immediately after major holidays or technical events.

This may mean that your schedule doesn't exactly match your template. That's ok, the template is a guide, not an absolute requirement.

Post your proposed schedule to the wiki, using the organization described at the start of Part III, or something similar.

A good practice is to add a change log to the wiki below the schedule. The schedule will inevitably change as the release cycle proceeds. Use the log to track changes to the schedule as well as TSC milestone approvals. This is a good way to be transparent about schedule changes. It's also helpful as a reference for the post-release retrospective.

The last step is to review the schedule with the TSC. TSC members may have information about scheduling that you do not. Therefore, you may need to iterate the schedule once or twice to capture the feedback and get approval. Remember to add the approval date to the schedule log.

Sub-Project Release Plans

THE LINUX FOUNDATION

LF NETWORKING

Sub-Project Release Plans

Recall that one of our goals with release management is to provide transparency to the community.

We've talked about creating a process and developing a schedule, but what do we know about what individual sub-projects are doing in the release? For example, what if I want to know the format that a particular sub-project is using for a particular software artifact?

This is where the sub-project release plan comes in. By asking that each sub-project create a plan, the project makes its goals and deliverables for the release transparent to the community.

To prepare for this, we need to develop a template that all of the sub-projects will use. This requires that we put together a proposal and socialize it with the project PTLs until we have a consensus.

Sub-Project Release Plans

So, what does a project release plan template look like? It can be as detailed and comprehensive as you'd like, but usually, simpler is better. We're trying to put as little burden on our sub-projects as possible, while still achieving the transparency that the community needs.

As mentioned in the introduction to Part III, the assumption is that release documentation is contained in the wiki. Perhaps the simplest way to do this with the release plan is to set up a template in the wiki, then have each project copy and paste it into their project wiki. Once the project has created a release plan once, they can copy and paste and update it for each subsequent release.

Here's an example from the Anuket project.

Sub-Project Release Plans

In order to make sure that we are meeting our goal of providing transparency to the community, we want to make sure that the release plans are well prepared. Therefore, we need to have a process for reviewing and approving the plans.

If there are a small number of sub-projects, it may be acceptable for the release manager to perform the review and provide feedback.

A much better approach is for a member, or members, of the community to do the review. This could be done by a standing sub-committee, or rotated among members of the TSC, etc.

Tracking Release Progress

THE LINUX FOUNDATION

LF NETWORKING

Tracking Release Progress

As we've said, one of the goals of the release process is to provide transparency to the community about the status of the release.

It should be obvious then that this implies that a member of the community should be able to easily review the status of a release independent of the release manager.

Recall that our release process is based upon a set of milestones, where each milestone is achieved through the completion of release management tasks.

Therefore, to show the status of a release, we need:

1. A milestone schedule
   - Gantt, table, etc. showing milestones and scheduled completion dates
2. Milestone and release management task status
   - Which tasks for which projects for which milestones have been completed

Tracking Release Progress

The process for a community member to determine release status is:

1. Look at the schedule to find the next milestone
   - Also, consult the schedule log to review any changes to the schedule
2. Look up the milestone in the milestone tracker (we'll talk more about this) and review the status of tasks for the milestone

In the previous section, we talked about creating a schedule. So, our focus now is really #2. How do we track milestone status?

Tracking Release Status

Recall that, in general, each project has a set of tasks to complete for a given milestone. When all projects have completed all of their tasks for the milestone, then we recommend to the TSC that the milestone be approved.

So, in order to show release status, we need to include:

1. The milestone
2. The projects
3. The release management tasks assigned to each project

We are going to look at two ways to do this:

1. Confluence tables
2. Jira

Tracking Release Status

Note about collecting release status information

This is somewhat redundant to the Roles and Responsibilities section in Part I. However, it bears repeating.

Project participants must bear the responsibility for reporting release status into the release tracking system.

We do not want to have a situation where the Release Managers must query every project about status for every release management task for every milestone.

For a project with a significant number of subprojects and release management tasks, this will quickly become untenable. Release managers will get burned out and will not want to continue in the role.

On occasion, release managers will need to reach out to projects to get information, but this should be the exception, not the norm.

Confluence Tables

One way to track release status is to set up tables in your wiki, inside of the release-specific section that you have set up. IOW - let's say our release is called "alpha", then we should have a section in our wiki devoted to alpha.

The idea is to create a separate table for each milestone. In each milestone table, place the sub-projects in the right-hand column and place the tasks in the top row (you could probably reverse this if it suits you).

To start out, put a red 'X' in each cell. As projects complete their tasks, change the red 'X' to a green check (alternatively use different fill colors). If a task is not applicable to a particular project, then grey out the cell, and or enter "N/A".

Once all of the red Xs have been turned to green checks, the milestone is complete.

Confluence Tables

Milestone 1	Task 1	Task 2	Task 3	Task 4
Project 1	X	X	X	✅
Project 2	N/A	✅	X	X
Project 3	X	X	✅	✅
Project 4	✅	X	X	N/A

Release status tracking example using Confluence tables.

Jira

A problem with using tables to track status is that it tends to only work for relatively simple situations, where there are a small number of projects, and the tasks are straightforward.

If there are a large number of projects with a significant number of complex tasks to complete for each milestone, then tables tend to break down.

One reason for this is that complex tasks often require explanations in order to track progress, or to understand status. If you try to add explanatory text to a table cell very often, then the table quickly becomes unwieldy and difficult to interpret, thereby losing transparency to the community.

This is where Jira can be useful. Jira enables much better tracking of complex information, including explanatory text, history, status reversals, etc.

Jira

Note: We're using Jira because Jira is common at the LF. However, it's also possible that this method could be adapted to another issue tracking tool.

Since we are already using Jira to track issues and bugs, we need a way to separate our release management tasks so that they are easily searchable.

One way to do this is to take advantage of Jira's ability to create issue hierarchies with an Epic.

So, for each sub-project, we create an Epic for each milestone. Then we create child tasks for each Epic that represent the release management tasks for that milestone (Example).

Jira

In order to be able to separate milestones from each other and from different release cycles, we can use a naming scheme for the Epic Name.

[release name] [milestone number] [milestone name]

Example: alpha M2 specification freeze

If we use this naming scheme consistently across all projects and all releases, then we will be able to easily search for the exact Epic that we want.

We also want to be able to separate out the release management tasks from other tasks that the community might want to associate with the Epic. For this, a Jira label is useful. I have frequently used the label "relman". It doesn't matter, as long as it is unique within your Jira instance.

Another nice feature of Jira is that it integrates with Confluence. This means that we can summarize milestone status information from Jira in tables and graphs in our wiki (example).

Jira

At this point, you may be thinking that this sounds like a lot of Jira issues to write! For example, if we have

25 sub-projects X 5 milestone tasks + 25 Epics

that's 150 Jira issues for a single milestone!

A couple of things can help.

First, you can take advantage of the Clone++ tool, which is available as a Jira plugin. This way, you can write an Epic and it's subtasks once, then clone them for every project. You may need to make some edits, such as deleting or closing tasks for projects where they are not applicable.

If you're feeling ambitious, another approach is to use the Jira API to script creation of the Jira issues. For example, Python has a plugin that works well with the Jira API. Once completed, the script can generate all of your issues in seconds.

Contact me for more information if you're interested in pursuing this route.

Confluence Tables vs. Jira

Use Confluence Tables if…

- Small number of sub-projects (< 10)
- Release management tasks are relatively simple

Use Jira if…

- Significant number of sub-projects (>= 10)
- Significant complexity in release management tasks
- Willingness and ability to write lots of Jira issues

Conducting Milestone Reviews

THE LINUX FOUNDATION

LF NETWORKING

Conducting Milestone Status Reviews

The role of the release manager in a milestone status review is to present the information to the TSC and to make a recommendation.

Ultimately, however, it is up to the community, via the TSC, to decide whether the milestone has been achieved.

Therefore, it is important to spend some time thinking about how to conduct a milestone review.

1. The milestone review should be added to the TSC agenda. If necessary, the release manager should request that the topic be added.
2. Shortly before the milestone date, the release manager should thoroughly review the status of all of the tasks and any other requirements for the milestone and decide on a recommendation
3. At the TSC meeting, the release manager should carefully review the data with the TSC and give their recommendation.

Conducting Milestone Status Reviews

4. The TSC and the release manager may then engage in a discussion and respond to any questions.

At this point, there are a couple of possibilities:

- The TSC agrees with the release manager's recommendation.
  - The TSC may wish to conduct a vote, or they may just ask for any objections to approval.
- The TSC disagrees with the release manager's recommendation
  - This may be due to information that a TSC member has, or that they simply disagree that the data backs up the recommendation.
  - This is perfectly valid and is no cause for the release manager to be concerned. It's part of the process.
- In either case, document the decision in the meeting minutes and update the schedule log.

Milestone Status Reviews

If the decision is that the milestone requirements have not been met, then the next step is to decide on a correction:

- Conditional approval. If a small number of release management tasks (< 10%) have not been completed, then the TSC may wish to grant a conditional approval to the milestone. So, for example, the unfinished tasks would need to be completed in a certain period of time, or by the next milestone.
- Adjust the schedule. This may involve all or just one of the milestones.
- Adjust the scope. If a requirement appears to be no longer feasible, then it may be necessary to adjust the scope of the release to change or remove the requirement.
- Project withdrawal. If a project is unable to complete release management tasks, then it may need to withdraw from the release.
- Exception. The TSC may wish to grant an exception to a project for a particular requirement. For example, a requirement to update the version of Python might be deferred to the next release. Note that exceptions should be documented and tracked.

Whatever the mediation, it should be formally approved by the TSC and clearly documented in the minutes, in detail.

Managing Exceptions

Managing Exceptions

- Inevitably, you will encounter a situation where a project is unable to meet a milestone requirement on time, or implement a release requirement.
- If this is limited to a single project, then it often doesn't make sense to adjust the release schedule, or the scope, for everyone to accommodate one project. However, we also don't want the project to withdraw from the release. So, what do we do?
- Often we can use an Exception Request. An exception request is a a request to the TSC to deviate from the planned schedule, or a release requirement, for a single participant.
- In order to be transparent, we need a way to track these. Often the simplest approach is to set up a table in the wiki with the other release-specific information.
- The table could include all of the information directly, or it may include a link to a form that documents the exception request (example).

Managing Exceptions

Here are some fields that you might want to track:

- Submitters Name
- Project or Requirement Name and JIRA
- Milestones affected
- Projects affected
- Background description
- Schedule impact
- Recovery plan
- Milestone schedule change
- Risk
- Status
- Decision

Managing Exceptions

The TSC must decide how they want to approve exception requests. Here are two approaches:

1. TSC (or delegate) reviews and approves each exception request. This works well if the number of requests is fairly small.
2. Exception request is automatically approved, unless there is an objection from a TSC member. This works well for large projects with a relatively large number of requests. The downside is that requests often don't get reviewed by TSC members and the community is heavily dependent on an "honor code" approach to avoid objectionable exception requests.

Release Retrospective

THE LINUX FOUNDATION

LF NETWORKING

Release Retrospective

- You may recall back in Part II, that I said that you should not overthink the development of the release process.
- The goal is to make a good effort to nail down the tasks and schedule template, then execute the process and iterate.
- In order to iterate, we need to be able to examine what went wrong and what could be improved.
- The goal of the release retrospective is to do just that. However, we need to have data to review.
- If you have a six-month cadence, a lot can happen over that period of time. When things go wrong, you may think that you'll never forget it, but you might be surprised how little you remember of the incident a few months later.

Release Retrospective

- In order to iterate and improve the release process, it is important that you capture and document issues that come up so that they can be addressed in the retrospective.
- The most effective way that I've found to do this is for the release manager to keep notes of issues that arise with as much detail as possible.
- These notes may be kept in a wiki table as the release proceeds, or they may be published once the release has been signed off (example).
- Share the notes with the community and invite them to supplement them with their own observations.
- Another source of data is the schedule log. For example, if you notice after multiple releases that you always have to slip a particular milestone date, that's a clue that you may need to adjust your schedule template.

Release Retrospective

- After you've allowed some time for the community to add to the notes, schedule time in a meeting for the retrospective. Rather than the TSC meeting, the retrospective should be conducted in a meeting with the technical team.
- Another approach is to conduct the retrospective as a session at a technical event (e.g. Dev & Test Forum).
- One approach to the retrospective is to simply review the list of notes accumulated by the release manager and the community. This is useful if you have a limited amount of time.
- Review each item, ask the author of the note to comment, invite discussion, then decide on a disposition for that item. The disposition might be to make a change to the process, or it may be to do nothing for now and gather more information in the next release.

Release Retrospective

- Another approach, which may be combined with the review of the notes, is to review the milestones, in order, using the schedule log as a guide.
- Pay particular attention to instances where a milestone was slipped, especially if it was slipped more than once.
- Often a review like this can jog memories and generate valuable discussion that can lead to important insights into the process. However, it does take time to complete.

Part IV: Next Steps

Next Steps

1. Use the established release management working group meeting as a forum to develop the release management process, using the guidance presented in this training.
2. Seek approval for the process from the TSC.
3. Develop a schedule for the first release cycle using the new release management process.
4. Seek approval from the TSC for the release schedule.

Questions?

THE LINUX FOUNDATION

LF NETWORKING