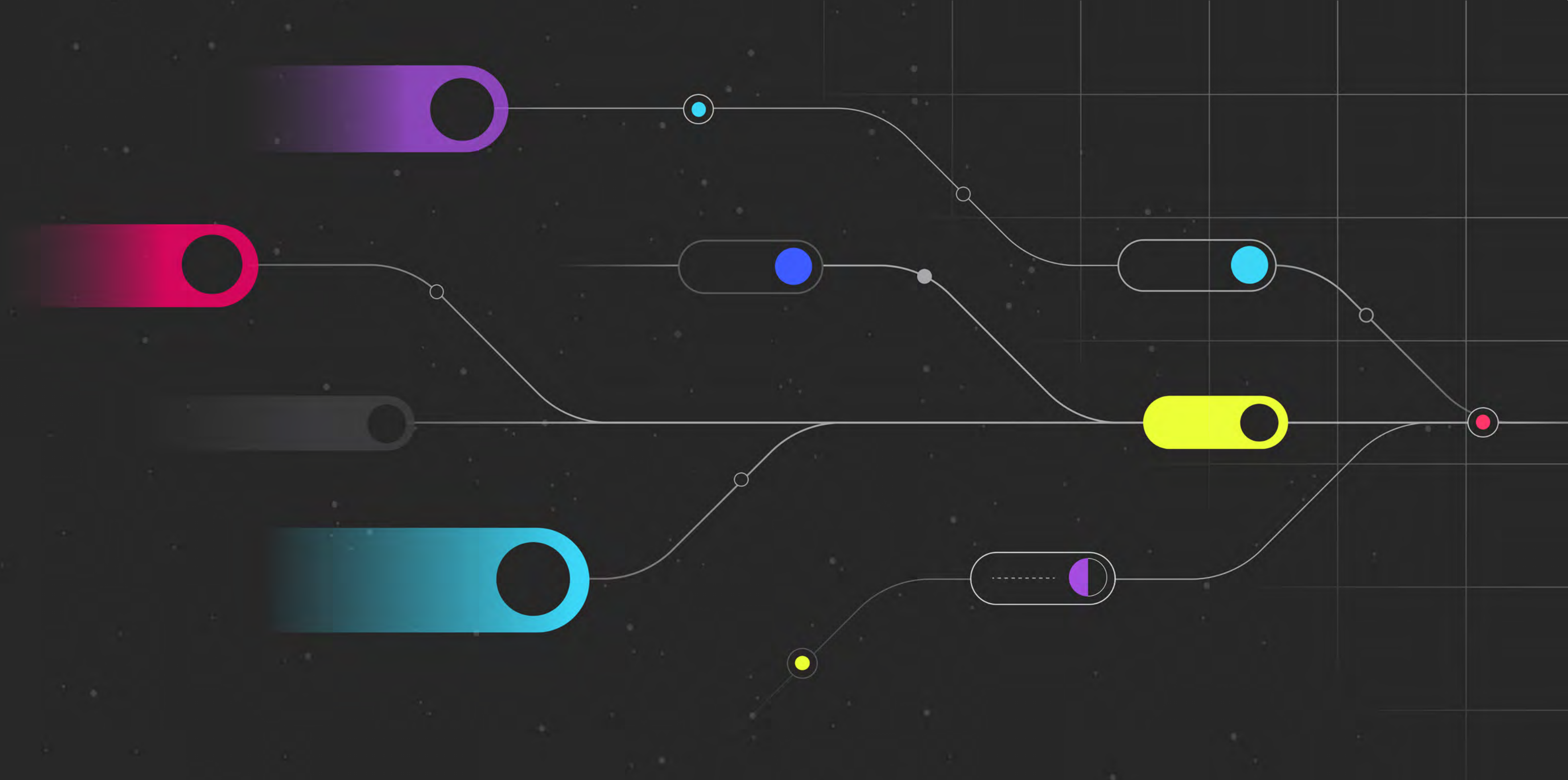


7 best practices for short-term and permanent flags

Tips and tricks, definitions, use cases, and more to improve your feature management workflow

LaunchDarkly

Contents

Introduction	02
Short-Term Flags	03
Permanent Flags	04
Best practices for all feature flags	05
Best practices for permanent feature flags	09
Best practices for short-term feature flags	10
1 best practice for leveraging best practices	12

Introduction

Feature flags, sometimes called feature toggles, are standard practice for many software development teams. As the top feature management platform, we're often asked for advice on feature flag best practices.

And what are those best practices? Well, it depends on

Whether the flag is a short-term or permanent flag. The purpose of the flag.
Your specific business needs. What works for one company may not work for others.

In this ebook, you'll read about the nuances of feature flagging best practices, so you can avoid technical debt and other common troubles and use feature flags to your benefit.

First, you need to determine whether a flag will be a short-term or permanent flag, as that will influence future decisions and best practices.

Short-term flags

A short-term flag has a limited lifespan, and you generally remove the flag once it has fulfilled its business purpose. Thinking of feature flags, most people think of short-term flags. Types of short-term flags include:

Release

Slowly exposing a feature to new users—moving from internal to beta and/or canary users until 100% of users receive the feature. When you reach 100%, remove the flag (unless it's needed as a circuit-breaker as described below).

Segment your population to determine a preference for one option over another. Once testing ends, remove the flag, and 100% of users should receive the preferred variation.

Operational interaction testing

When rolling out a new microservice, infrastructure component, or third-party tag, a flag can be used to determine the impact on systems. If the CPU spikes or there is a memory leak, or unexpected errors occur, disable the new element while further troubleshooting takes place.

Maybe you're thinking that a release flag sounds a lot like an operational interaction testing flag. They are similar, but the primary difference is the controller. An operational interaction flag is controlled by the ops team to protect the systems, whereas the release flag is controlled by the product or business owner and controls how user adoption progresses.

Permanent flags

A permanent flag is designed to provide control for an extended time after the release of a given feature. In some cases, the flag will be in existence for the life of a feature.

Circuit breakers/Load shedding

Having the ability to quickly turn a feature off or terminate a connection when problems arise prevents problems from impacting all users. These flags are often activated based on an event. For example, a monitoring tool generates an alert when orders fail to complete. When the alert is triggered, a flag is toggled, setting the site to "read-only."

White labeling

Configuring the look and feel of an application for each client for a white-labeled solution.

Best practices for all feature flags

Whether you have a permanent or short-term flag, consider these best practices. Please note: These are recommendations that we follow and have collected from other customers. Recommendations may change over time. As such, feel free to modify and alter based on your specific needs. What works well for one company or team may not work well for another team.

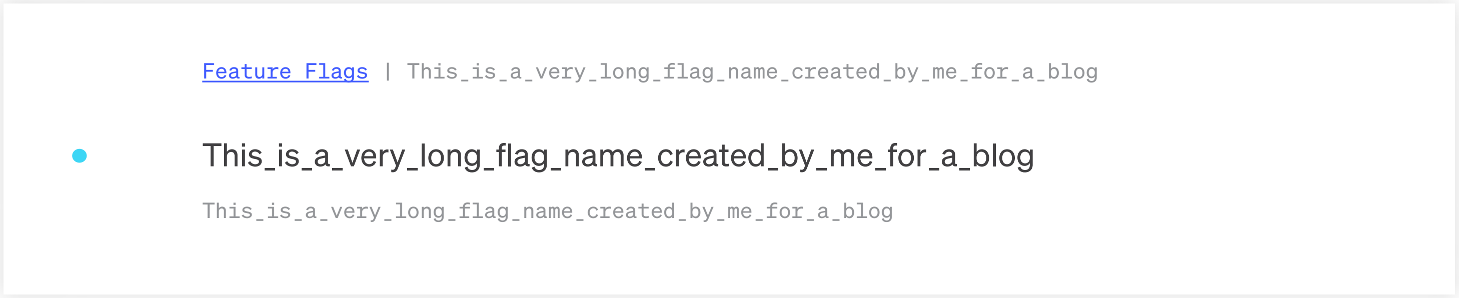
1 Make flag planning a part of feature design

Feature flags shouldn't be an afterthought. If you think about feature flags during the design process, you will be setting yourself up for success. Once you decide whether the flag will be a short-term or permanent flag, this decision will then impact other areas such as a naming convention, configuration settings, review and removal processes, and access control and safety checks. We suggest proper planning upfront for all flags.

2 Standardize naming

You may have a style guide outlining conventions on how to write code for your application; this could include things like when and where to use CamelCase or the proper use of indentation. These style conventions make it easier to read and understand the code.

Before creating your first flag, come up with a naming convention to be used. Our first recommendation is for verbose flag names. Don't try to be brief. Verbose flag names can help others understand what the flag does.



Feature Flags | This_is_a_very_long_flag_name_created_by_me_for_a_blog

This_is_a_very_long_flag_name_created_by_me_for_a_blog

This_is_a_very_long_flag_name_created_by_me_for_a_blog

Things to consider when writing the style guide or naming convention:

- Be descriptive about the flag's behavior.
- Include a prefix with a project name or team name.
- Indicate whether the flag is temporary or permanent.
- Include a creation date for the flag. (This will be helpful when cleaning up flags, more on this below).

Whether or not to use flag in the name: When using a service like LaunchDarkly, using flag in the name is redundant. If you're using a home-grown solution, using flag in the name may help clarify the purpose of the code.

For example, suppose you are creating a flag to progressively roll out and test a new chatbox widget of your UI. This will be a short-term flag. Without a naming convention in place, you may end up with a flag called “brand-new-flag” or “new-UI-widget.”

These names don’t tell us a whole lot about the flag. But with a standard naming convention in place that addresses all of the above, you can create a more descriptive flag name like “aTeam-chatbox-widget-temp-030619.” We know from the name that this is a temporary flag for a chatbox widget created by the “a team” on June 3—much better!

3 Minimize the reach of a flag

The focus of a flag should be small. Having a flag that controls more than one feature action at a time can be confusing and will make troubleshooting issues harder. Think about the smallest unit of logic needed for the most responsive flag. If there are multiple parts to a feature that have to work together, we suggest creating a master flag as a dependency.

For example, say you’re launching a new dashboard in your application. The dashboard has three widgets. You should create a total of four flags: one flag for each widget with a dependency on a fourth flag for the main dashboard. With this scenario, if one widget causes problems, the dashboard with two widgets can still be served.

4 Review use at regular intervals

Whether creating a short-term or permanent flag, you need to review flag use at regular intervals. The frequency at which you review the flags may vary based on business requirements and the type of flag. To avoid accrual of technical debt, check both permanent and short-term flags at a regular cadence.

For short-term flags, look to see if the flag has rolled out to 100% of users or if a flag is served to no users. For permanent flags, examine whether the flag is still needed (was a feature once part of a premium bundle but is now available for all users?) We will cover specific criteria related to removing short-term flags below.

5 Establish access control and safety checks

If you have regularly scheduled flag clean-up events, you may worry about the accidental removal of permanent flags. Minimize this risk by implementing access control and safety checks.

Within LaunchDarkly, a flag cannot be deleted without confirmation, but that is a partial solution. There are two additional ways to implement:

1. Use tags and custom roles to assign permissions to flags within LaunchDarkly quickly.

2. Set role-based access control (RBAC) to specify who can delete flags in a given environment.

Best practices for short-term feature flags

1 Create a process for removing flags before you create one

Coding a flag is a two-part process. The act of removing a flag should not be a separate process from the act of creating a flag. As mentioned above, you should plan for flags during the feature design process. This includes the removal of short-term flags. An easy way to handle the removal process is to write a pull request to remove the flag at the time you create it.

Schedule a GitHub reminder for after the feature is deployed to review and determine if the PR to remove the flag should be committed.

2 Conduct regular clean-up and review cycles

Avoiding and eliminating technical debt is necessary. Here are some ways to schedule flag clean-up.

- Schedule time at the end of every sprint to review existing flags.
- Perform a clean-up/refactoring sprint at a regular cadence (quarterly, semi-annually, whatever works for your business) to pull out old flags & tags.
- Make it a competition. Hold a “Capture the Flag” day. The individual or team that removes the most flags wins.

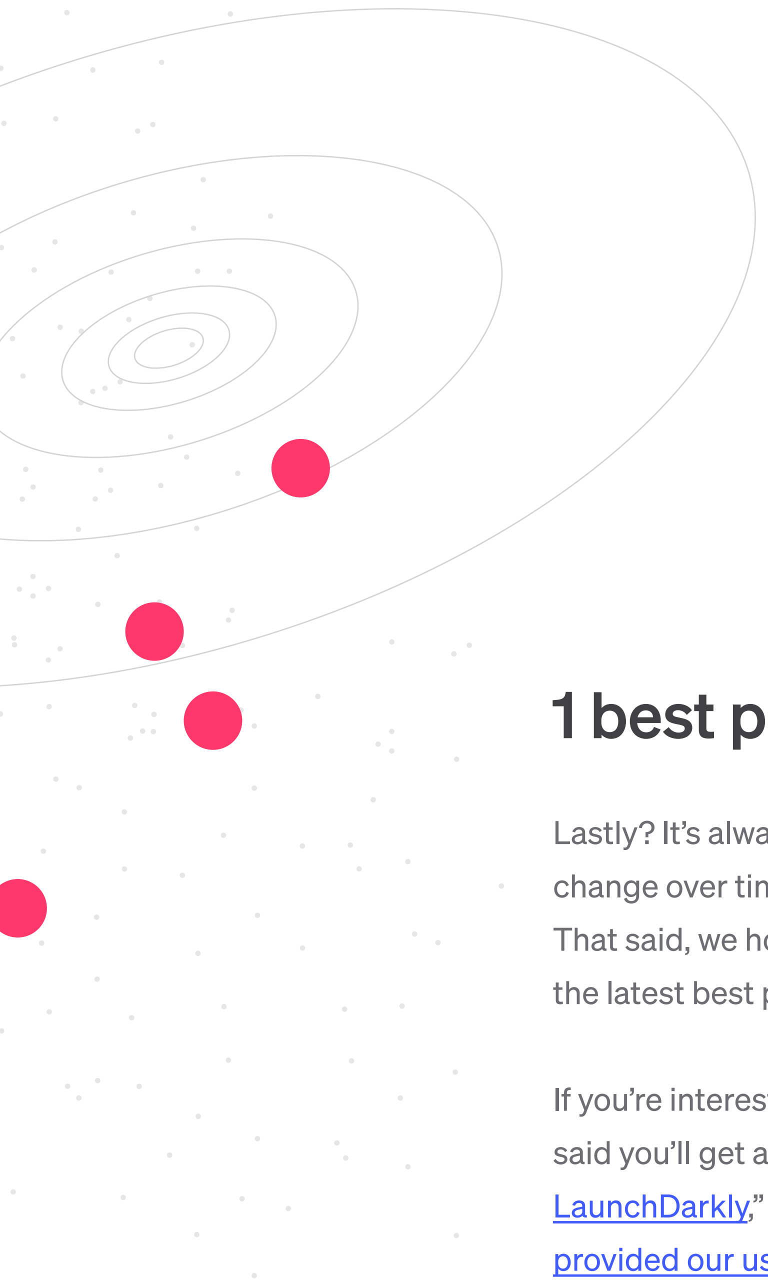
Within LaunchDarkly, we make it easier to identify flags for removal.

You can:

- Filter tags by creating a date to view the oldest flags
- View which variations of a flag were recently served. Are 100% of users receiving the same variation?
- Filter on the last evaluation date of a flag

Once you have identified a flag for removal, use code references to find all the instances of that flag in your codebase.

If you are not using LaunchDarkly, a consistent naming strategy can help you prep the code for instances of flags.



1 best practice for leveraging best practices

Lastly? It's always a best practice to modify any advice for your team. Recommendations may change over time, and what works well for one company or team may not work well for another. That said, we hope you've found these tips helpful and if you're interested in staying updated on the latest best practice, check out the LaunchDarkly blog.

If you're interested in taking the plunge with LaunchDarkly, see why consulting firm Forrester said you'll get a 245% ROI from our platform in their report, "The Total Economic Impact of LaunchDarkly," or go here to view survey results and learn more about the ROI LaunchDarkly has provided our users.

Empowering all teams to deliver and control their software.

launchdarkly.com

sales@launchdarkly.com

LaunchDarkly