Tips for managing your feature flags (and your technical debt)

Feature flags are great. Even better: Maintaining them.

You probably know by now that feature flags promise faster releases, controlled rollouts, and efficient experimentation. But you won't get very far without a system for keeping your flags tidy and up-to-date.

Here are some best practices to help you get there. We'll look at ways to make sure you're able to maximize your feature flags throughout their life cycle.

Note: Even if you don't have a feature flag solution yet, you can still play around with some of these practices by trying out Unleash's live demo. It's free, open, and super fun to use.

Set up strong naming conventions

Most feature flag management solutions allow you to name your toggles whatever you want. It's your choice whether or not you create naming conventions. Generally, it's a good idea.

How you name your toggles is up to you. A good idea is to include key information as part of a flag's name.

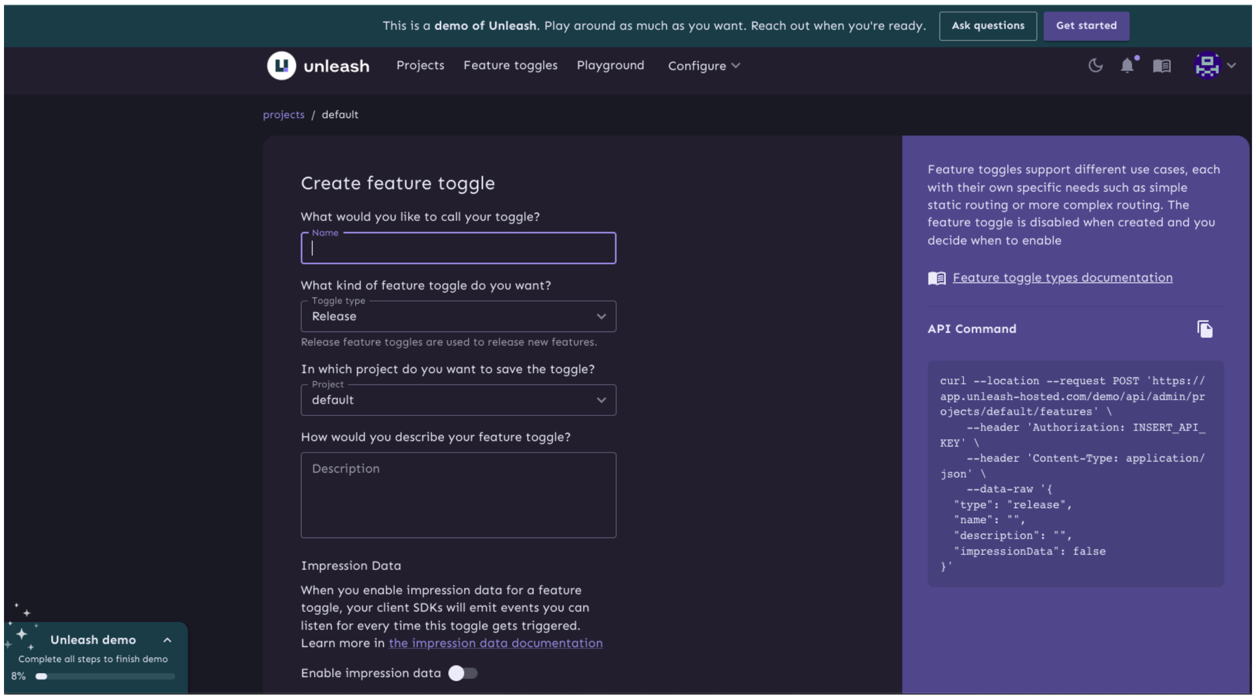
For example:

- "[TicketId]"- for example JiraID
- The name of the impacted service or functionality
- Toggle type - for example, "Experimental," "GradualRollout," or "TimeBased"

The goal is clarity. You should avoid getting too technical with your naming conventions so that both developers and project managers know what the flag is supposed to do.

It's also a good idea to document your naming conventions. Over time this can become a guideline for future use cases. Conventions don't mean much if they're not consistent.

Naming your toggles in Unleash is simple. You can also choose the type of toggle, and, if it makes sense, a description of what the toggle is for.



This is a **demo of Unleash**. Play around as much as you want. Reach out when you're ready.

Create feature toggle

What would you like to call your toggle?

Name:

What kind of feature toggle do you want?

Toggle type: Release

Release feature toggles are used to release new features.

In which project do you want to save the toggle?

Project: default

How would you describe your feature toggle?

Description

Impression Data

When you enable impression data for a feature toggle, your client SDKs will emit events you can listen for every time this toggle gets triggered. Learn more in the impression data documentation

Enable impression data

Feature toggles support different use cases, each with their own specific needs such as simple static routing or more complex routing. The feature toggle is disabled when created and you decide when to enable

Feature toggle types documentation

API Command

```
curl --location --request POST 'https://
app.unleash-hosted.com/demo/api/admin/pr
ojects/default/features' \
    --header 'Authorization: INSERT_API_
KEY' \
    --header 'Content-Type: application/
json' \
    --data-raw '{
  "type": "release",
  "name": "",
  "description": "",
  "impressionData": false
}'
```

Focus on communication, Clear and descriptive names make it easier for developers and stakeholders to understand the purpose of each flag.

Make the scope of your toggles clear and consistent

You won't find many restrictions on what functions you use your toggles for. Again, how you organize is up to you.

In this case, it's a good idea to make sure you create a toggle specifically for an individual function, and that's it. This can help keep unintended behaviors to a minimum.

This also means you should avoid creating multiple toggles for the same function. It's just easier to manage this way. Plus it avoids confusion.

Keep your existing route live while your toggle is disabled

When your toggle is active, it should represent a new function. When it's inactive, it should represent the existing function.

For example, let's say you create a new payment gateway. Enabling the toggle should give your users the option to pay through the new gateway. A disabled toggle should point users to the existing gateway payment options.

In practice, your code might look like this:

```
// Define a list of available payment gateways
paymentGateways = ["PayPal", "Authorize.Net"]
newPaymentGateways = [“Stripe]

If toggleEnabled(toggleName.enableNewPaymentGateways):
  paymentGateways.extend(newPaymentGateways)

// Display the list of payment gateways in the UI
displayPaymentGateways(paymentGateways)

function displayPaymentGateways(gateways):
    // Loop through each gateway in the list
    for gateway in gateways:
        // Display the gateway name in the UI
        displayGatewayName(gateway)

        // Add a button or link for the user to select the gateway
        displaySelectButton(gateway)
```

Streamline and automate your release processes

Most feature flag management solutions are designed to make it easy to create and manage feature toggles, usually through a console.

It’s good practice to pair the solution with a release process. The idea is to effectively manage both your toggles and their states.

Let’s look at Unleash. A common method for programmatically managing feature toggles is using Unleash’s Admin API. It lets you perform operations like creating, updating, and deleting toggles using the level of automation that makes sense for your team.

You’re also able to maintain the toggle in a repository, along with its project, environment, and other configurations. A powerful approach could include defining a pipeline that a). reads the configurations from the repository, and b). uses Admin API to interact with the Unleash instance.

With this pipeline setup, you can automate a number of processes like updating feature toggles, managing states, and synchronizing them with the Unleash instance. This way you can streamline your feature toggle management workflow, and incorporate it into your software development processes.

A sample configuration file could look like this:

```yaml
---
version: 1
projects:
- id: test
  action: deleteIfExists   # Key to indicate if the project needs to be deleted if exists already
- id: default
  name: Default
  action: createIfNotExists  # Key to indicate if the project needs to be created if not exist already
  description: Default project
  features:
  - environments:
    - displayName: Development
      enabled: false
      name: development
    - displayName: Production
      enabled: false
      name: production
    name: demo
    stale: false
    type: release
    action: createIfNotExists
  - environments:
    - displayName: Development
      enabled: false
      name: development
    - displayName: Production
      enabled: false
      name: production
    name: demo.test
    stale: false
    type: release
    action: createIfNotExists
- id: MyNewProject
  name: MyNewProject
  action: createIfNotExists
  description: A test project
  features:
  - environments:
    - displayName: Development
      enabled: false
      name: development
    - displayName: Production
      enabled: false
      name: production
    name: demo
    stale: false
    type: release
    action: createIfNotExists
  - environments:
    - displayName: Development
      enabled: false
      name: development
    - displayName: Production
      enabled: false
      name: production
    name: demo.test
    stale: false
    type: release
    action: createIfNotExists
```

You could, for example, set up the pipeline to read the sample configuration, then replicate it across different environments
You might want to bring flexibility to the pipeline so you can quickly disable a toggle, in case you run into unexpected behavior.

A way to introduce flexibility could be to expedite the rollout of recent changes with a single click. You could also grant super admin privileges to toggle states directly in your user interface.

Use access control for better stability and security

Speaking of admin privileges, you might want to consider access control if you're working in a larger team. Team cultures can vary broadly. Regardless, it can get messy when everyone on a team can perform any action on a toggle.

Access privileges are great for organizations that prioritize security, stability, and audits.

If you're using Unleash, you'll find a built-in set of standard roles that can make granting access control smooth. You can also define custom project-specific roles if you need to grant fine-grained access privileges.

Track changes to your feature toggles

Usually feature toggles or flags are a part of a larger CI/CD process. That doesn't mean they shouldn't be traceable.

When you make a log of feature toggle changes, you create a reviewable historical record. This is awesome for understanding how each toggle evolved, as well as how changes have impacted the performance of a feature.

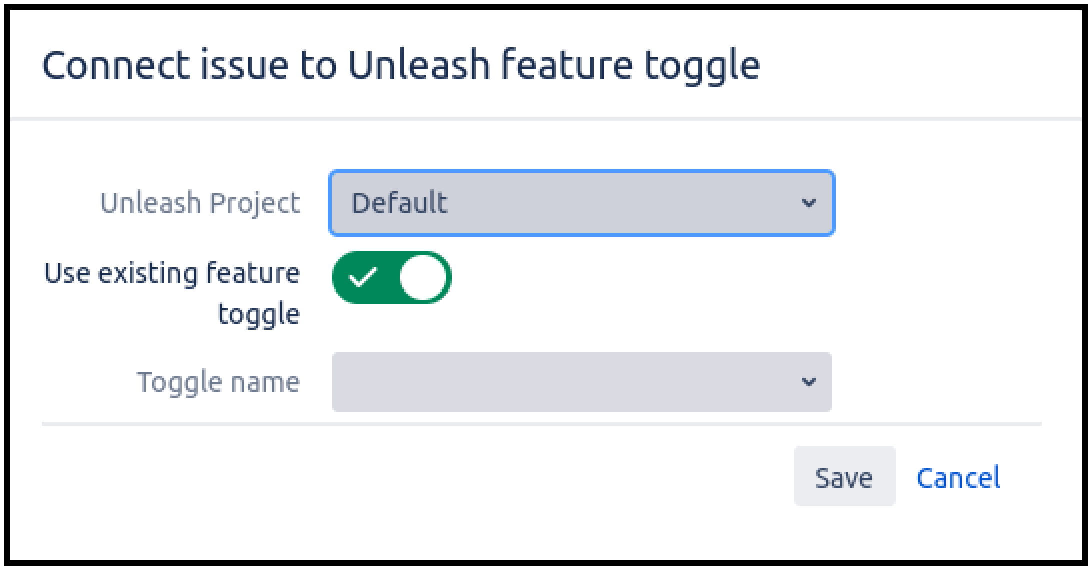
If you're using Unleash, you'll find that each feature toggle mutation is treated as an event:

- feature-created
- feature-updated
- feature-metadata-updated
- feature-project-change
- feature-archived
- feature-revived
- feature-strategy-update
- feature-strategy-add
- feature-strategy-remove
- feature-stale-on
- feature-stale-off
- feature-environment-enabled
- feature-environment-disabled

You'll be able to track changes through supported platforms like Slack, DataDog, and Microsoft Teams.

Unleash also exposes a webhook. You can use it to subscribe to the events you're interested in and set up event triggers for customized actions. s.

Plus, there's Jira. Unleash works with Jira to both create new toggles and link to existing toggles.


Connect issue to Unleash feature toggle

Unleash Project: Default

Use existing feature toggle: ✓

Toggle name:

Jira users with the right admin privileges can use Jira view and edit a toggle state. This is a great way to give project managers visibility into feature rollouts, as well as monitor and track feature toggles.

Don’t forget to clean up your loose feature flags

Once you’re confident that you won’t roll back a feature, you should remove its toggle.

If you leave unused toggles where they are, you’ll continue to build up the complexity with each new feature you release. This adds a lot of maintenance overhead and, really, technical debt. As a general rule, your feature toggle life cycle should be short.

Usually, that means after a short period of monitoring the toggle in production–like for, say, a few days or weeks. You should then remove the toggle from both the codebase and the client.

To keep your team honest, a great practice is to raise two tickets when you need a new toggle:

1. Create the feature toggle
2. Disable/delete the feature toggle

-

Make sure the toggle is also removed from the code. This is in case the same toggle name is used in the future.

Conclusion

Keeping your toggles clean and organized will make sure you get the most out of your feature flag management solution.

These are some of the best industry practices, but they're not all of them. You might identify your own on the way to your next feature release. You also might discover more by engaging in Unleash's Slack community.

Really, if you want to find out how great feature management can be for your releases, experiments, and overall software experience, these tips are a fantastic way to get there.