Feature Flags:
Simplify Branching & Deployments

Benjamin Day
TRAINER | COACH | DEVELOPER

@benday www.benday.com

Overview

- Limits of releases based on source control
- Feature flags
- Feature flag demos

Source control can sometimes get in your way.

Software development is hard.

Software delivery is hard.

How do you get a feature into production?

Most teams use a source control-based release flow.

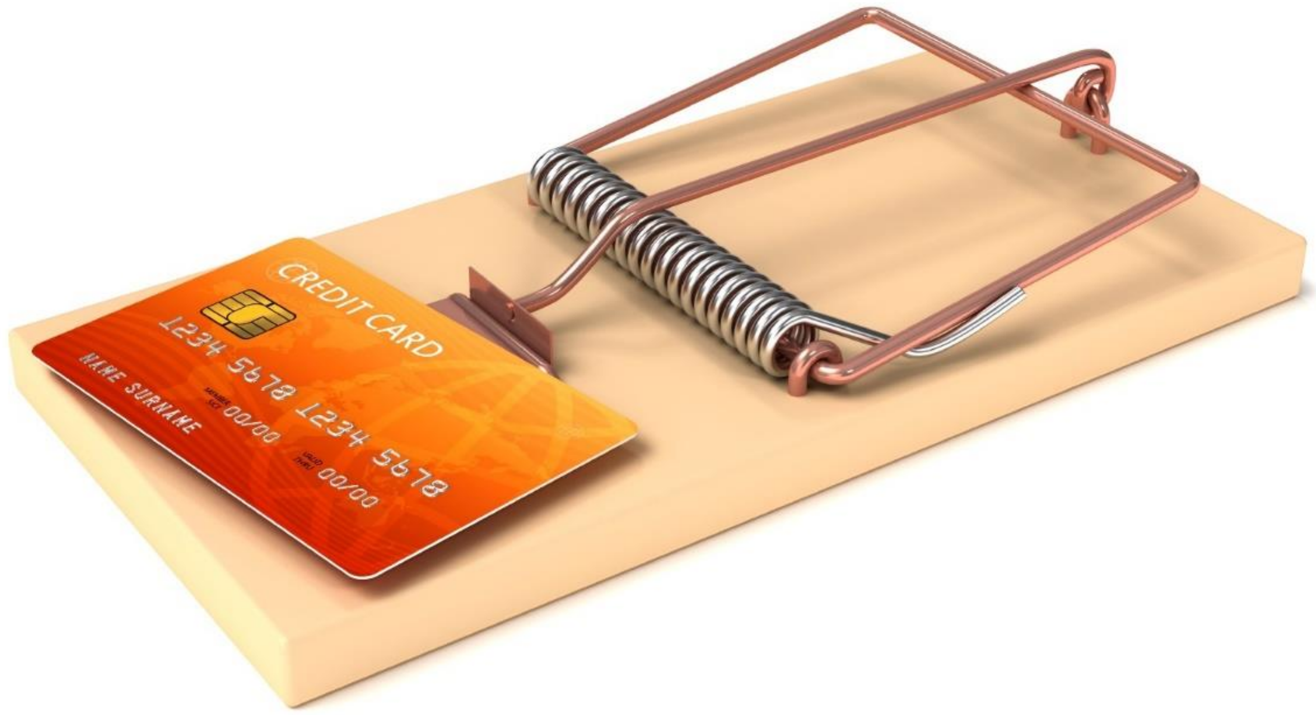
Branching & Merging

Branching lets you work on similar, related code in isolation
at the same time.

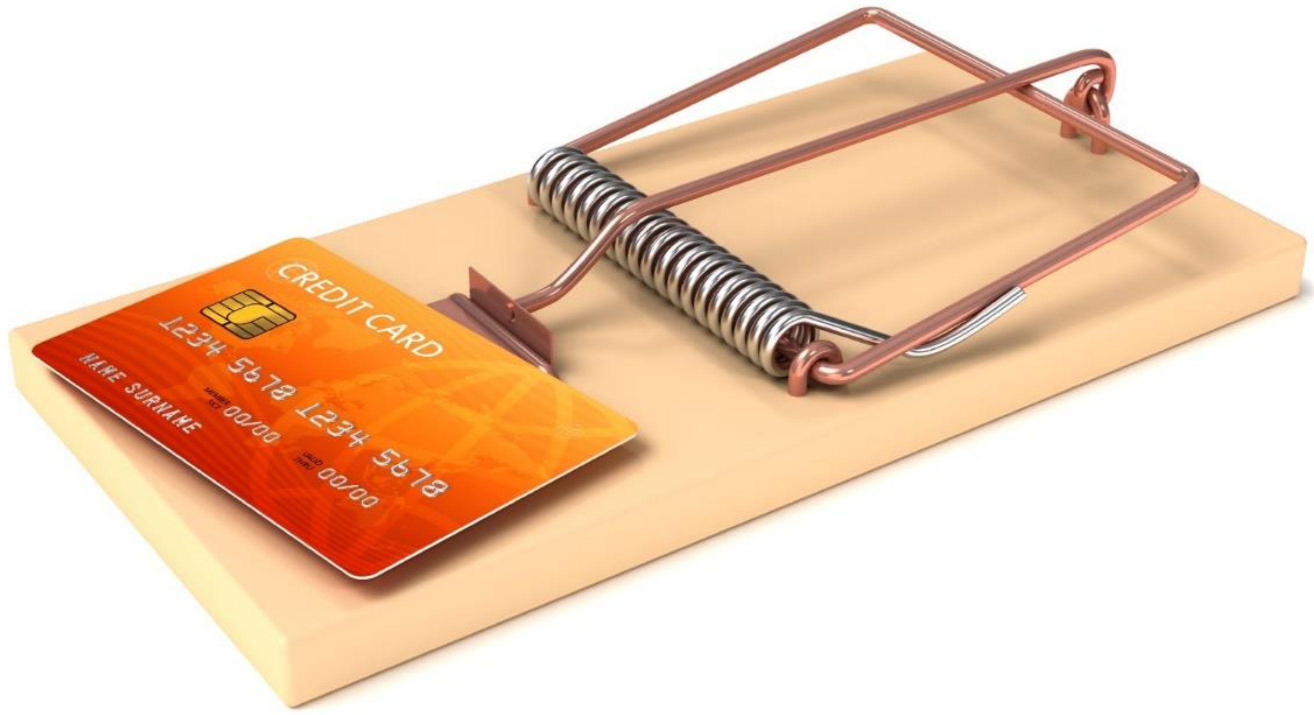
"So, I can go nuts and create 250 zillion branches and it's a good idea?"

No.

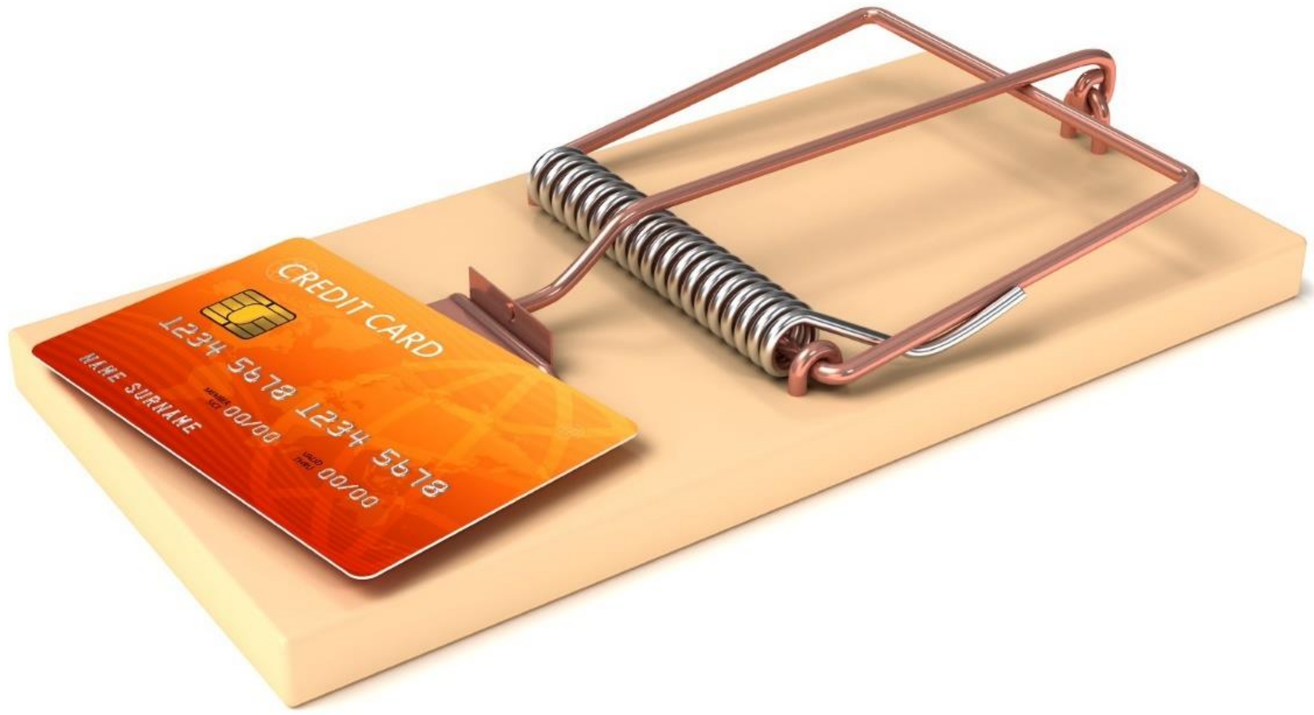
A branch is an integration credit card.



For every branch, there’s a merge.



Merging can be expensive and painful.



Until everything's integrated, it's *definitely* not done.

Keep it simple.

Best Practice: Don’t Branch

* - unless you absolutely have to

Integrate often.

The smaller the integration,
the easier it is.

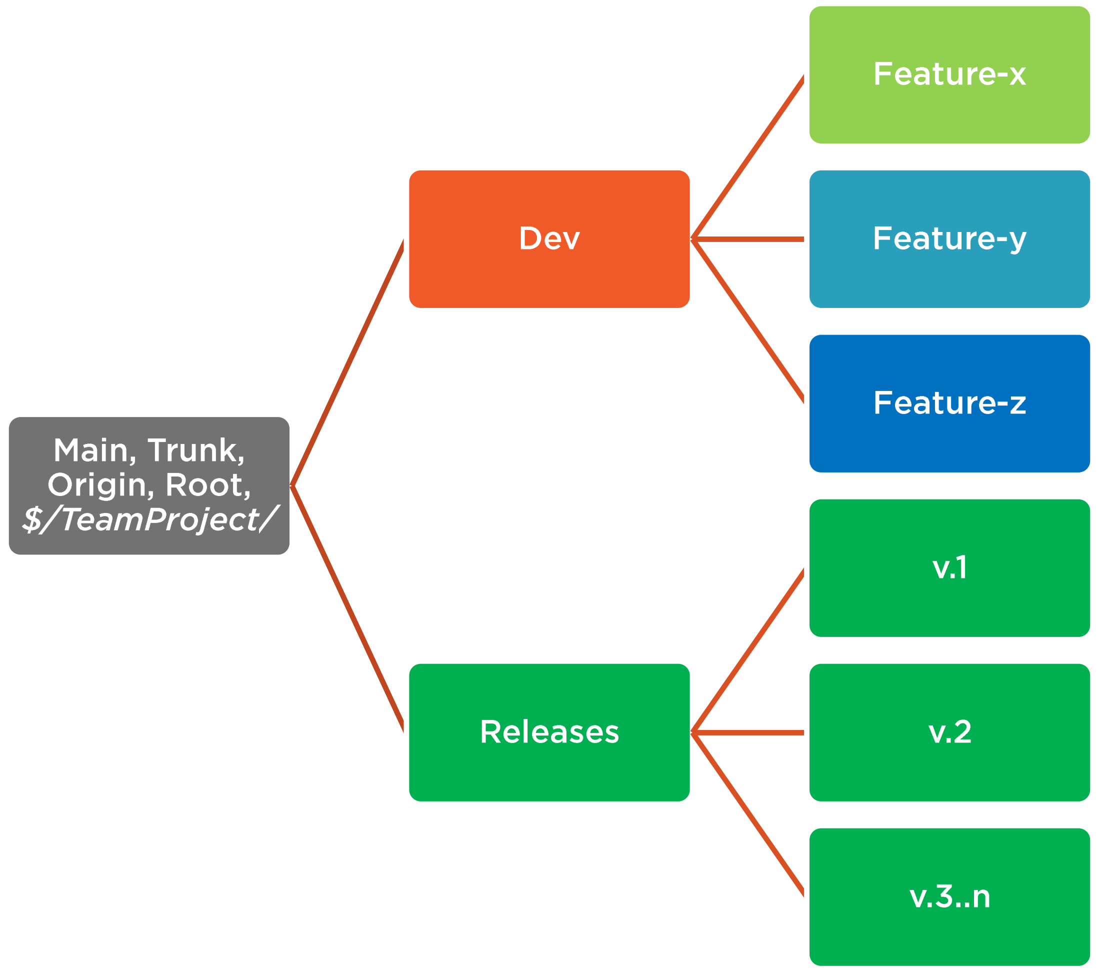
Review:
DevOps is about streamlined, automated flows.

Typical Branching Structure Activites

Branch for Development

Branch for Release & Release Maintenance

Typical Branching Structure



Main

- Beginning of everything
- Ultimate integration point

Dev

- Integration point for a release

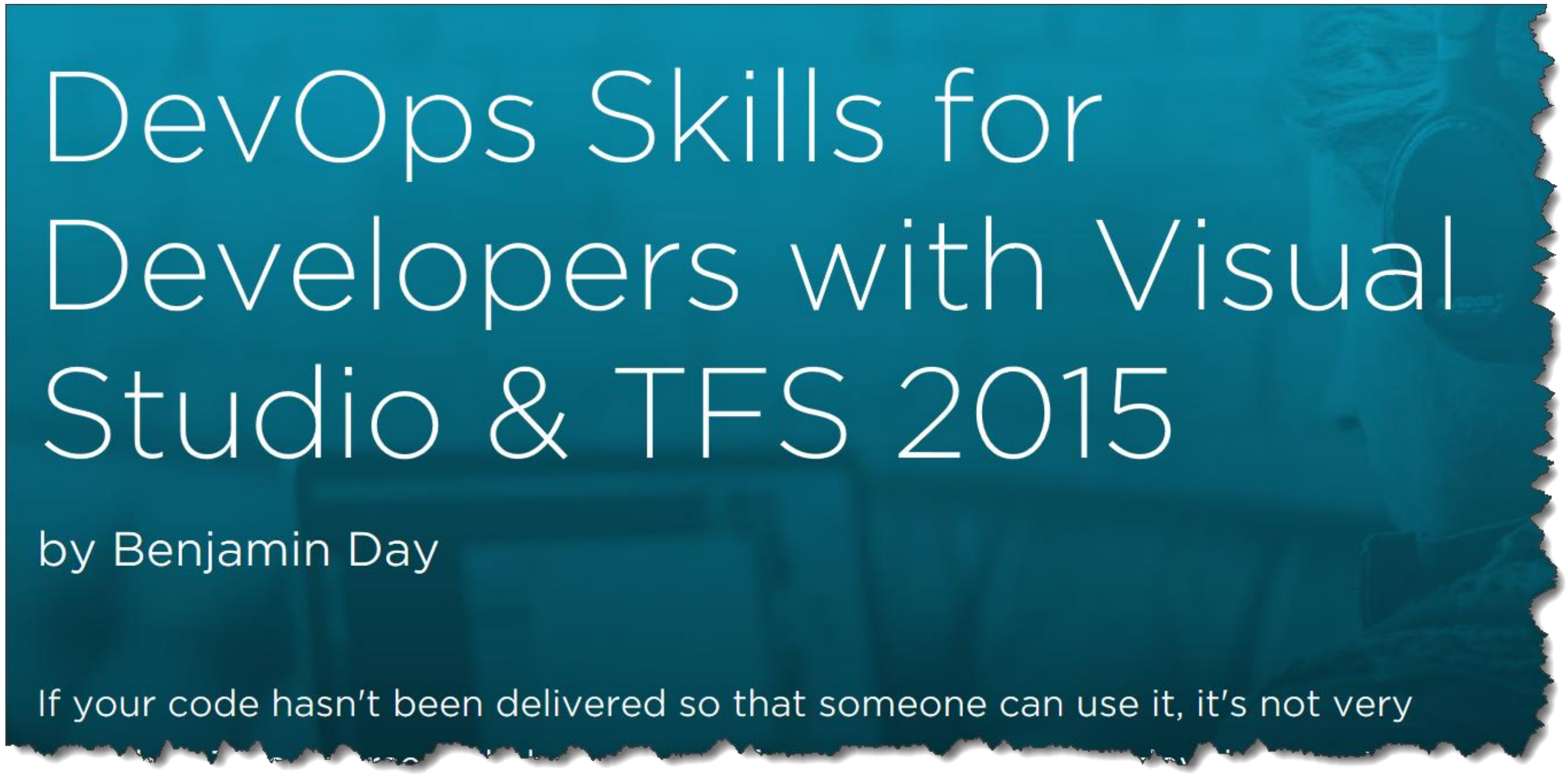
Features

- Features are developed here

Releases (v.*)

- Snapshot of what goes to production
- Source of hot fixes
- Using Git? Use tagging

Git Tags + TFS Release Branch Demos


DevOps Skills for Developers with Visual Studio & TFS 2015

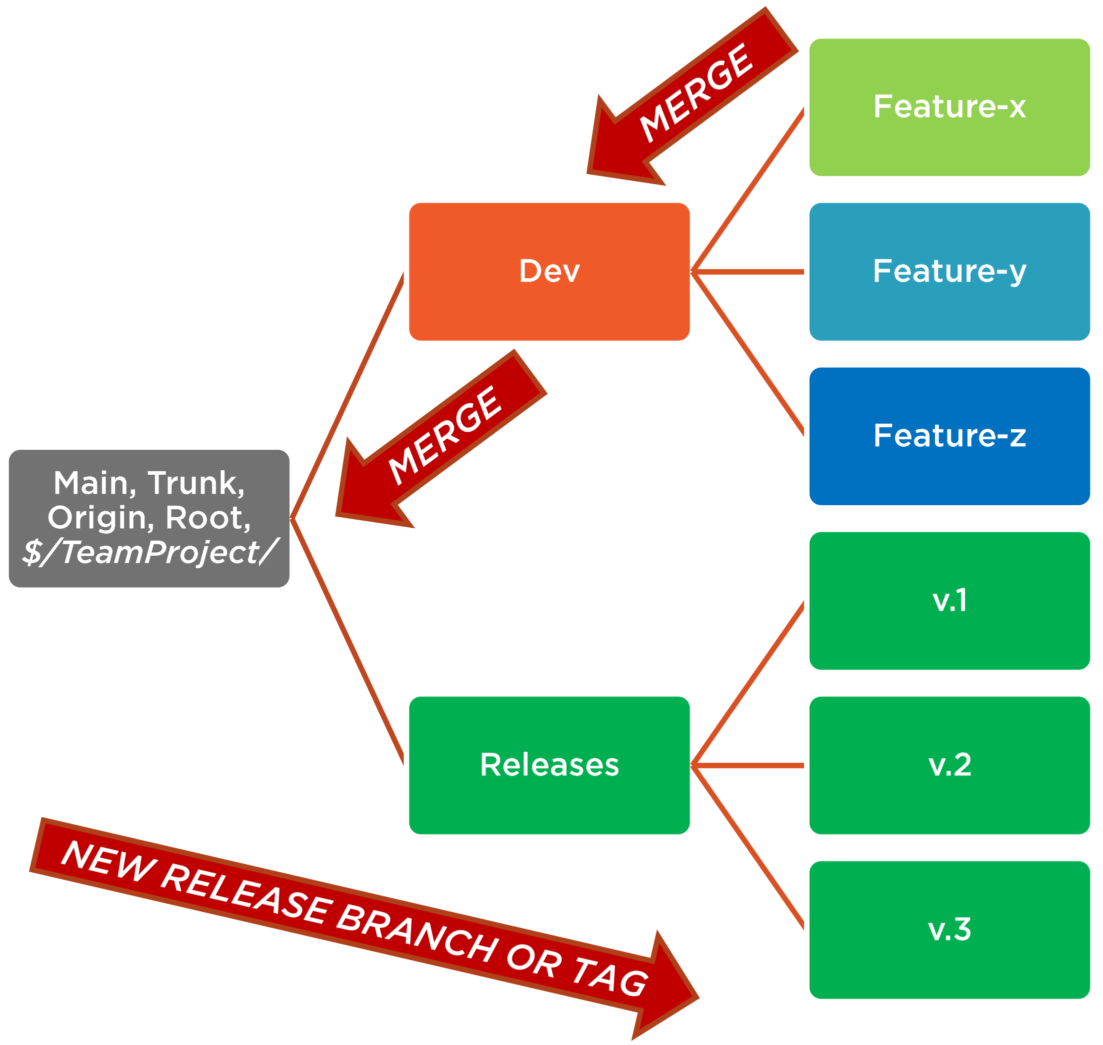
by Benjamin Day

If your code hasn't been delivered so that someone can use it, it's not very

"Managing 'Hot Fixes' & Code Quality: Branches & Code Reviews"

https://app.pluralsight.com/library/courses/devops-skills-developers-visual-studio-tfs-2015

Typical Release Flow


MERGE
Feature-x
Dev
Feature-y
MERGE
Feature-z
Main, Trunk, Origin, Root, *$/TeamProject/*
v.1
Releases
v.2
NEW RELEASE BRANCH OR TAG
v.3

Merge Features to Dev

Verify / fix whatever is in Dev

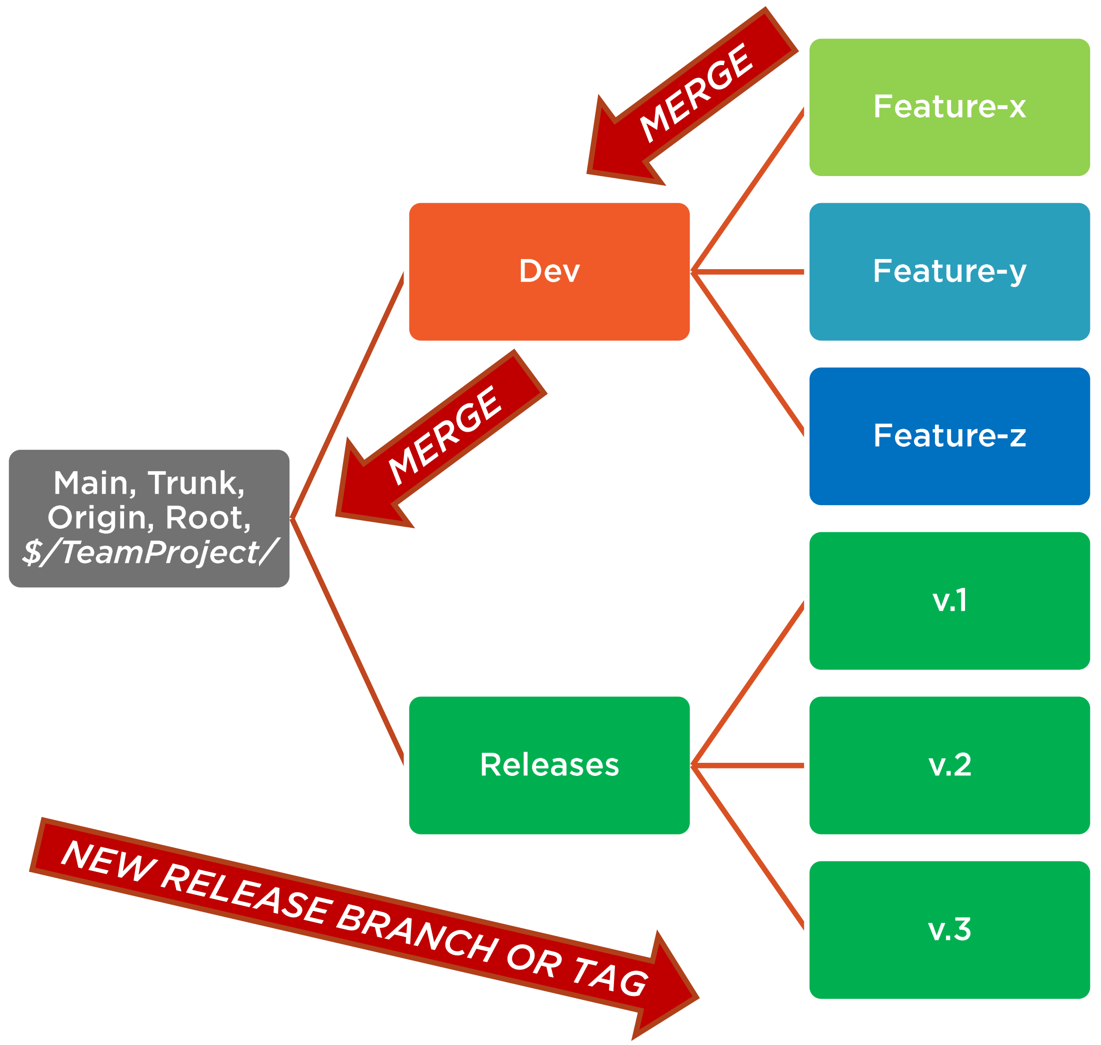
Merge to Main

Create new Release branch or Tag

- V.1123414
- 2016.03.15

Deploy to production servers

Typical Release Flow


MERGE
Feature-x
Dev
Feature-y
MERGE
Feature-z
Main, Trunk, Origin, Root, *$/TeamProject/*
v.1
Releases
v.2
NEW RELEASE BRANCH OR TAG
v.3

For a release, Main rules the universe

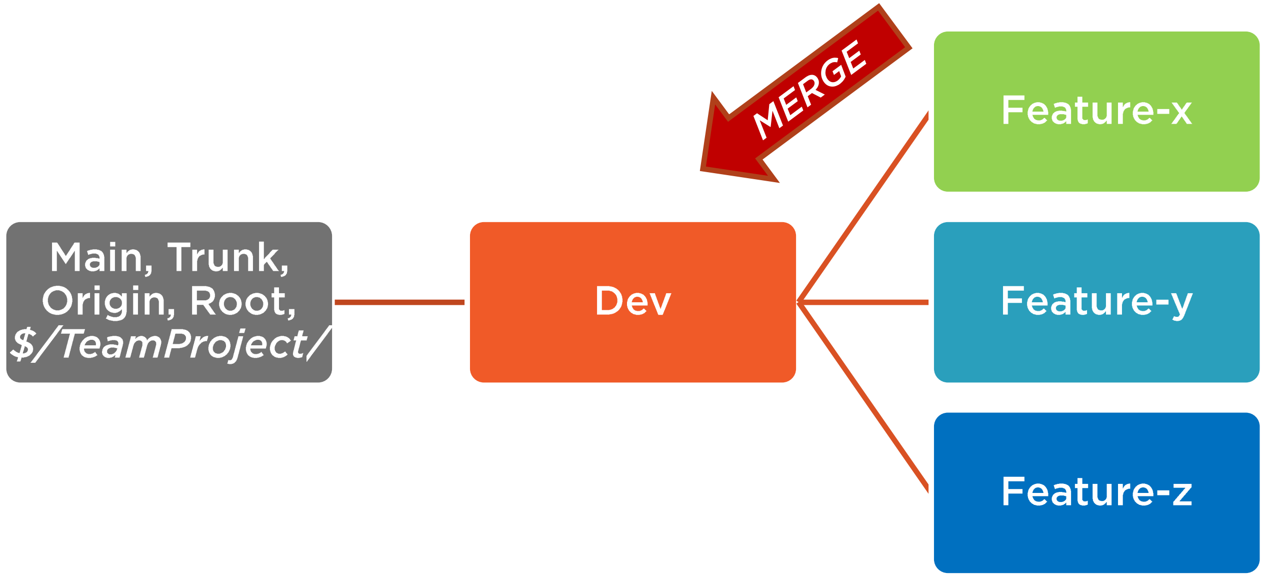
Whatever gets merged to Main gets released

Code-based / Source control-based release model

Next up:
How Source Control-based Releases Break

Source Control-based Releases Break

Typical Release Flow: Bad News



Merge Feature X

- Easy

Commit the merge

Merge Feature Y

- Less easy. Still ok.

Commit the merge

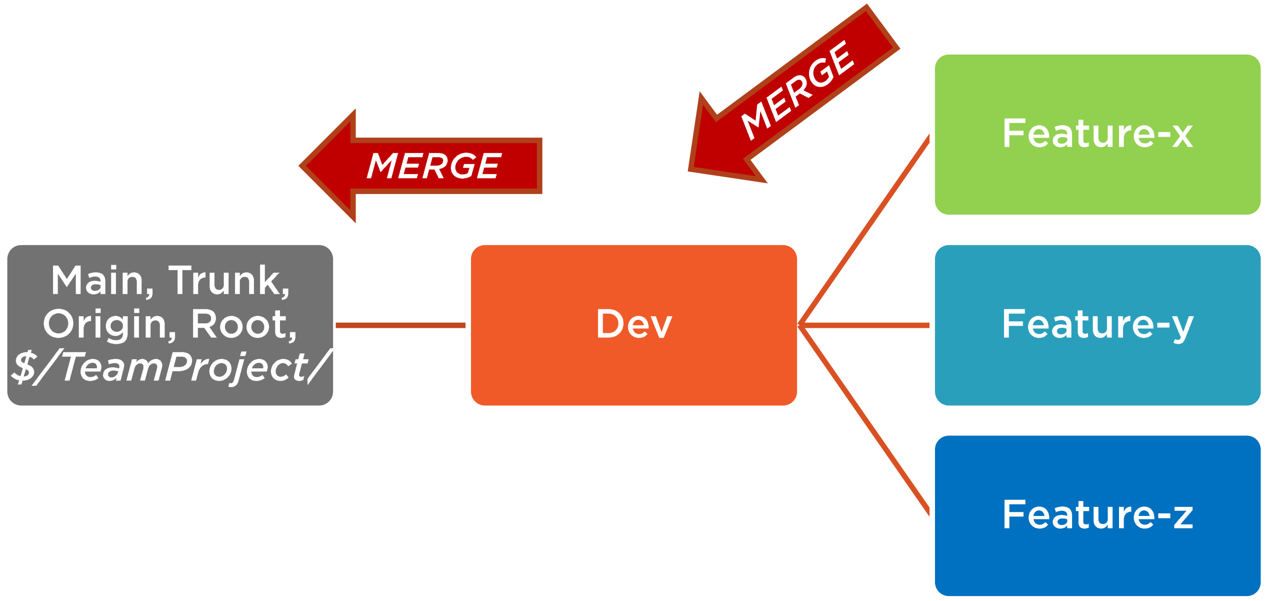
Merge Feature Z

- REALLY REALLY HARD!!

Turns out that Feature X wasn’t really done

Major surgery to get Feature X un-done

Typical Release Flow: Change of Plans



Merge Features X, Y, and Z to Dev

Commit the merge

Boss wants only Feature X and Feature Z to go live

“Feature Y should be in the next release.”

Merging Feature X and Feature Z to Main is painful, annoying, and tedious

Takes a long time

Delays

Commit the merge

Deploy to production…late

Reminder:
DevOps is about streamlined, automated flows.

Releases Based on Source Control Can Be a Real Drag

Whatever is in the branch goes to production

Whatever is in production is live

Pros:

- Simple to understand

Cons

- Coordination can be a real headache

Lots of manual effort

Next up:
Feature Flags

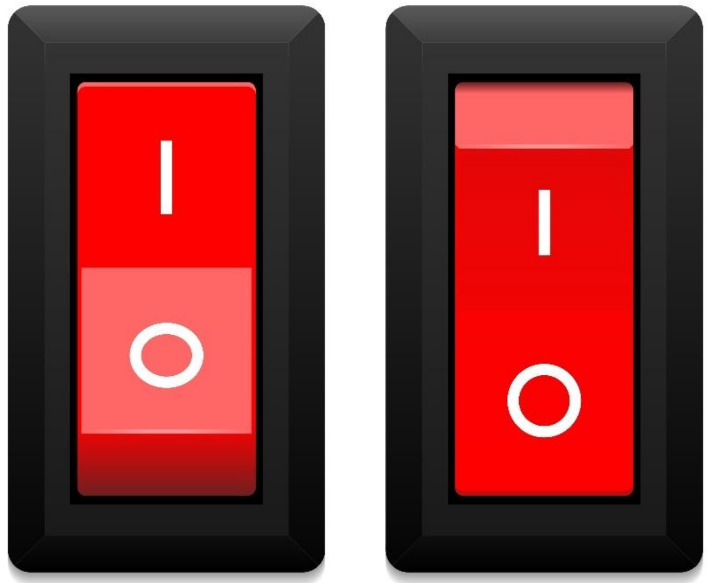
Feature Flags

Feature Toggles

Feature Flags

=

Configuration-based Releases



Feature Toggles / Feature Flags

Each new feature gets a configuration value

- Feature Flag
- On / Off

Wrap feature code in check for feature flag

- If it's on, it runs

Deploy whatever doesn't cause problems

- Closer to "deploy everything"

Turn on whatever is ready

Feature Flag Release Benefits

Decreased focus on branch/merge

Less deployment panic

- More flexible
- Easier to schedule

Deploy multiple version of a feature side-by-side

- New UI and old UI

Private betas

- New features for small subsets of users

A/B Testing

- Which version do users prefer?

Easy rollback

Feature Flag Drawbacks

Feature Flag Drawbacks

Adds complexity to the code

Can get confusing

Technical debt

Each feature flag should be removed

- Old versions of features need to be cleaned up

Can be an invitation to ship garbage

- Lack of focus on “Done”

Feature Flag Implementation Tips

Use Dependency Injection

Create an IFeatureManager interface

Each flag should be a Boolean property

- No magic strings!

No Magic Strings!

Wrong Way: Magic Strings

```
public class TheWrongWay
{
    private IFeatureManager _FeatureManager;

    0 references
    public void FeatureFlagsTheWrongWay()
    {
        if (_FeatureManager.IsFeatureEnabled("FeatureXyz") == true)
        {
            RunFeatureXyz();
        }
    }

    1 reference
    private void RunFeatureXyz()...
}
```

Right Way: Boolean Property

```
public class TheRightWay
{
    private IFeatureManager _FeatureManager;

    0 references
    public void FeatureFlagsTheRightWay()
    {
        if (_FeatureManager.FeatureXyz == true)
        {
            RunFeatureXyz();
        }
    }

    1 reference
    private void RunFeatureXyz()...
}
```

Feature Flag Implementation Tips: No Magic Strings

Each flag should be a Boolean property

Each flag should be temporary

Each flag should eventually be removed

Properties are much easier to find/remove

- Remove the property
- Recompile
- Fix the compile errors

Next up:
Feature Flag Demos

Demo

ASP.NET MVC Core

Basic feature flag implementation

IFeatureManager interface

Turn a feature on/off

Next up:
Two versions of the same feature at the same time

Demo

ASP.NET MVC Core

Add new functionality to an existing feature

Two versions of the same feature

Next up:
Private beta

Demo

ASP.NET MVC Core

Private Beta using Feature Flags

Turn on a feature for specific user(s)

Summary

Limits of releases based on source control

Feature flags

Feature flag demos

Next up:
Deploying applications using TFS Releases