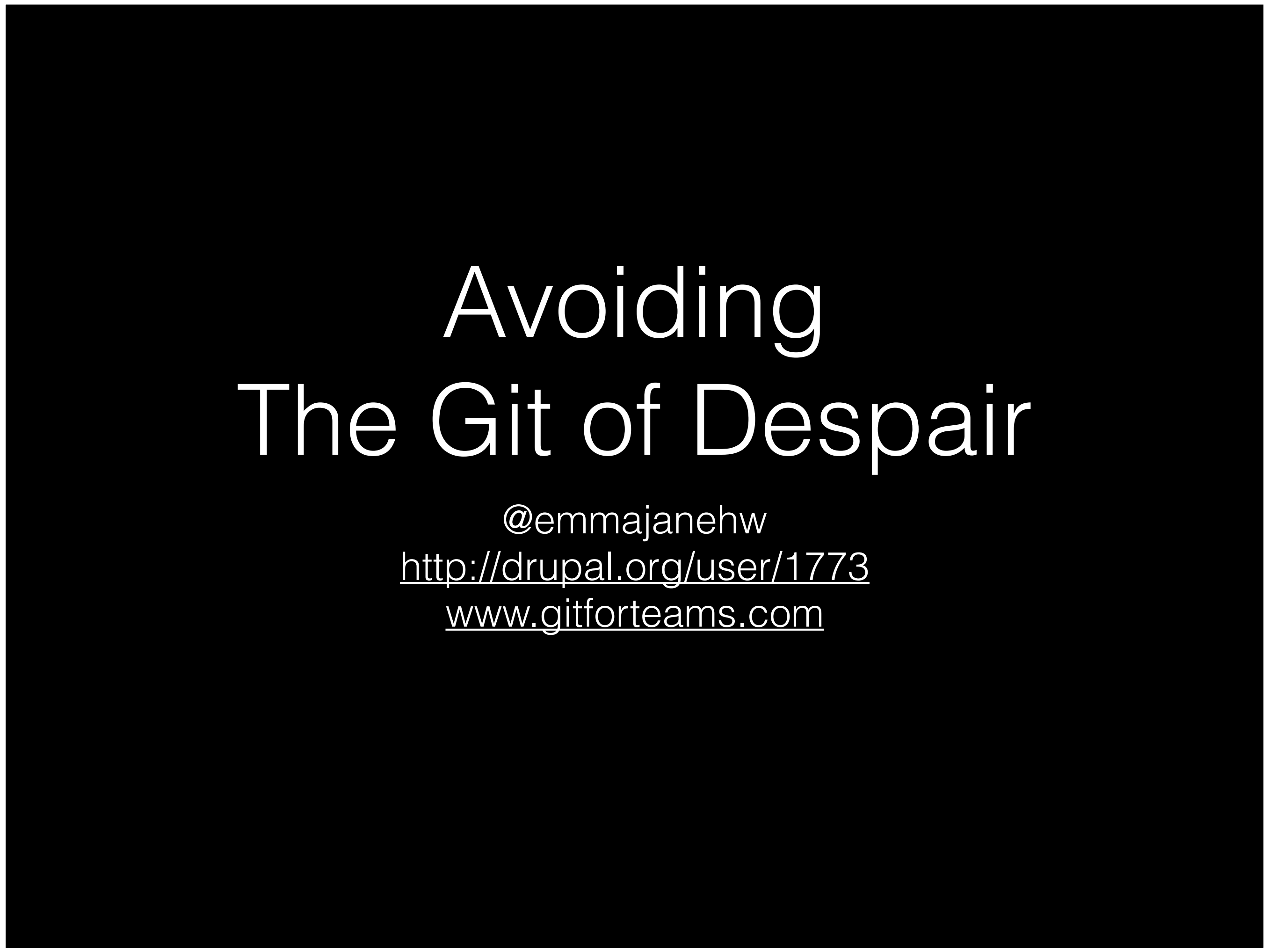
DRUPALCON ★ LOS ANGELES

# AVOIDING
## THE GIT OF DESPAIR

EMMA JANE HOGBIN WESTBY

SITE BUILDING TRACK

@EMMAJANEHW
http://drupal.org/user/1773



# Avoiding
# The Git of Despair

@emmajanehw
http://drupal.org/user/1773
www.gitforteams.com

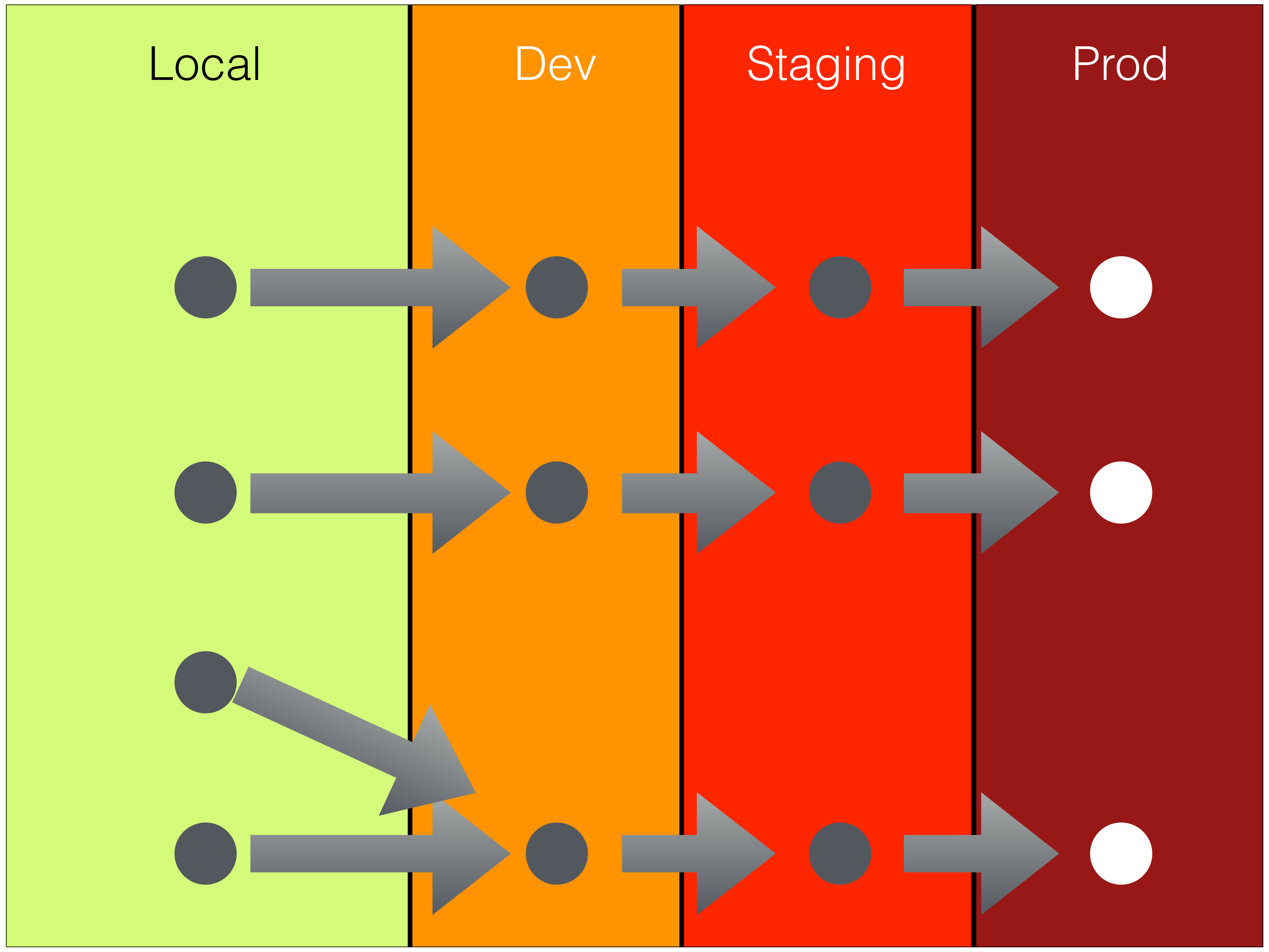
Back end developers have it easy. Just miles and miles of text-based code. Site builders, on the other hand, have to rely on point-and-click exportables from CTools / Features. Let's face it, there's a lot of magic going on, and it doesn't always go well. In this session we'll explore where all that stuff *goes* so that you're not constantly tripping over yourself in Git.

More specifically:

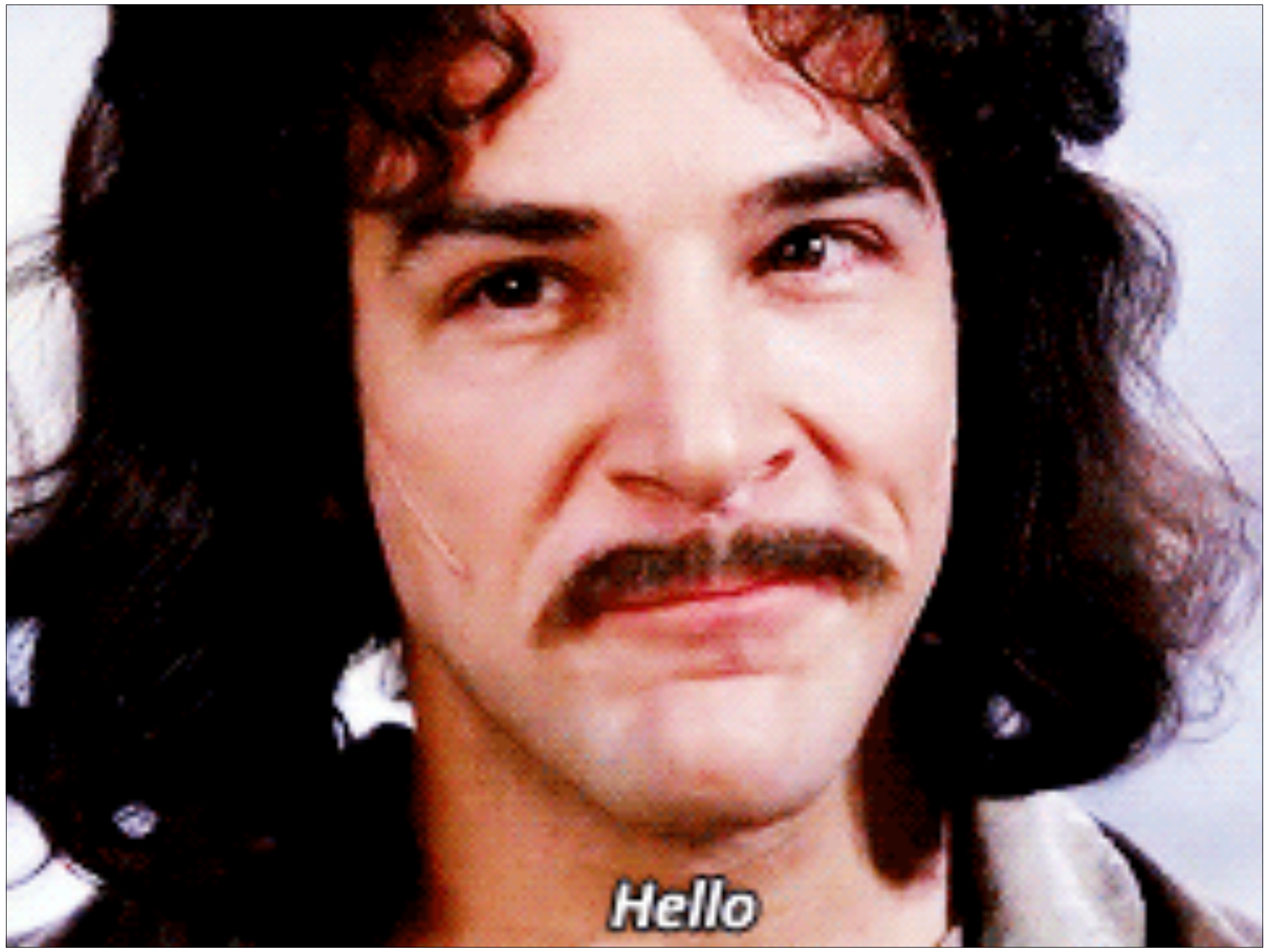
10,000ft view of how Git works with a deployment landscape (dev/stage/prod)
5,000ft view of how branches work, and what to do in Git world before you export a Feature
on-the-ground look at the commands you'll need to run once a Feature is exported so you can share it with others
5,000ft view of why you don't want to work on the same feature as someone else if you can avoid it
on-the-ground look at Feature-related merge conflicts just in case it happens to you
By the end of this session, you should be able to:

Describe a basic Git branching strategy, and how it relates to a typical dev/stage/prod infrastructure
Recall the Git commands you need to add a new Feature to your Git repository
Describe what a Git conflict is and how to recover from it
Throughout this session we'll explore the problems you might run into when working with Drupal-generated configuration files and Git. Although we'll focus on Drupal 7 and the Features module, the high level concepts will apply equally to other modules (and versions of Drupal!)

DrupalCon Los Angeles


Local
Dev
Staging
Prod

With Apologies
To Those Who’ve Not Seen
The Princess Bride.



Hello — I'm Emma. I've been working with Drupal + version control for over a decade.

In this presentation I'm going to give you an overview of how to use Git to push configuration-as-code to the right environment.

I don't want to say that this is easy stuff…because you won't believe me.

## Happening

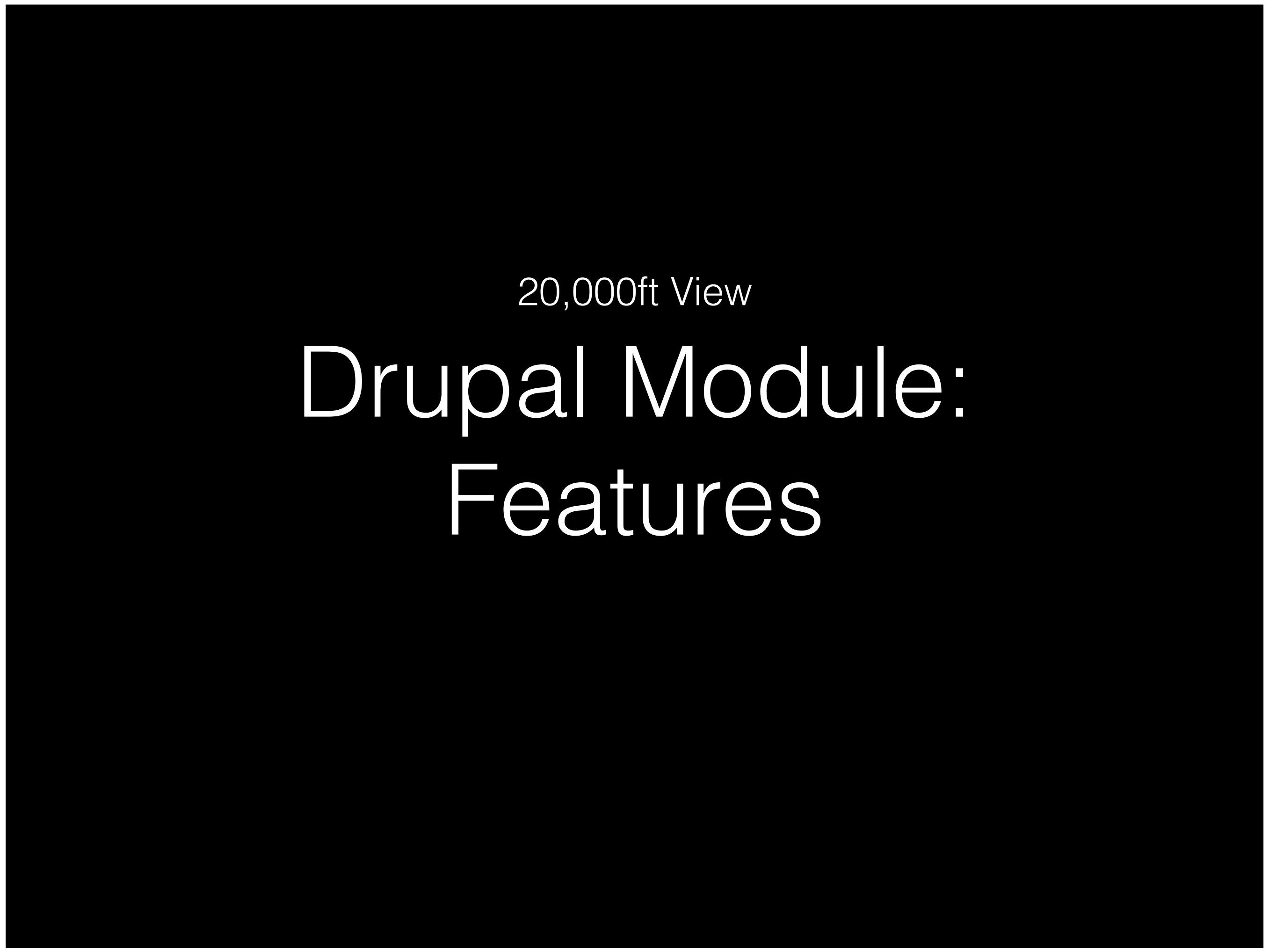
- How Git works in a deployment landscape.
- How to use branches for different environments.
- Commands you need to run.
- Why it’s hard to collaborate on Features.
- Commands to deal with with merge conflicts

## Not Happening

First-timer’s guide to:

- Drupal Module: Features
- Git (at the Command line)

But don’t run away just yet!



20,000ft View

## Drupal Module: Features

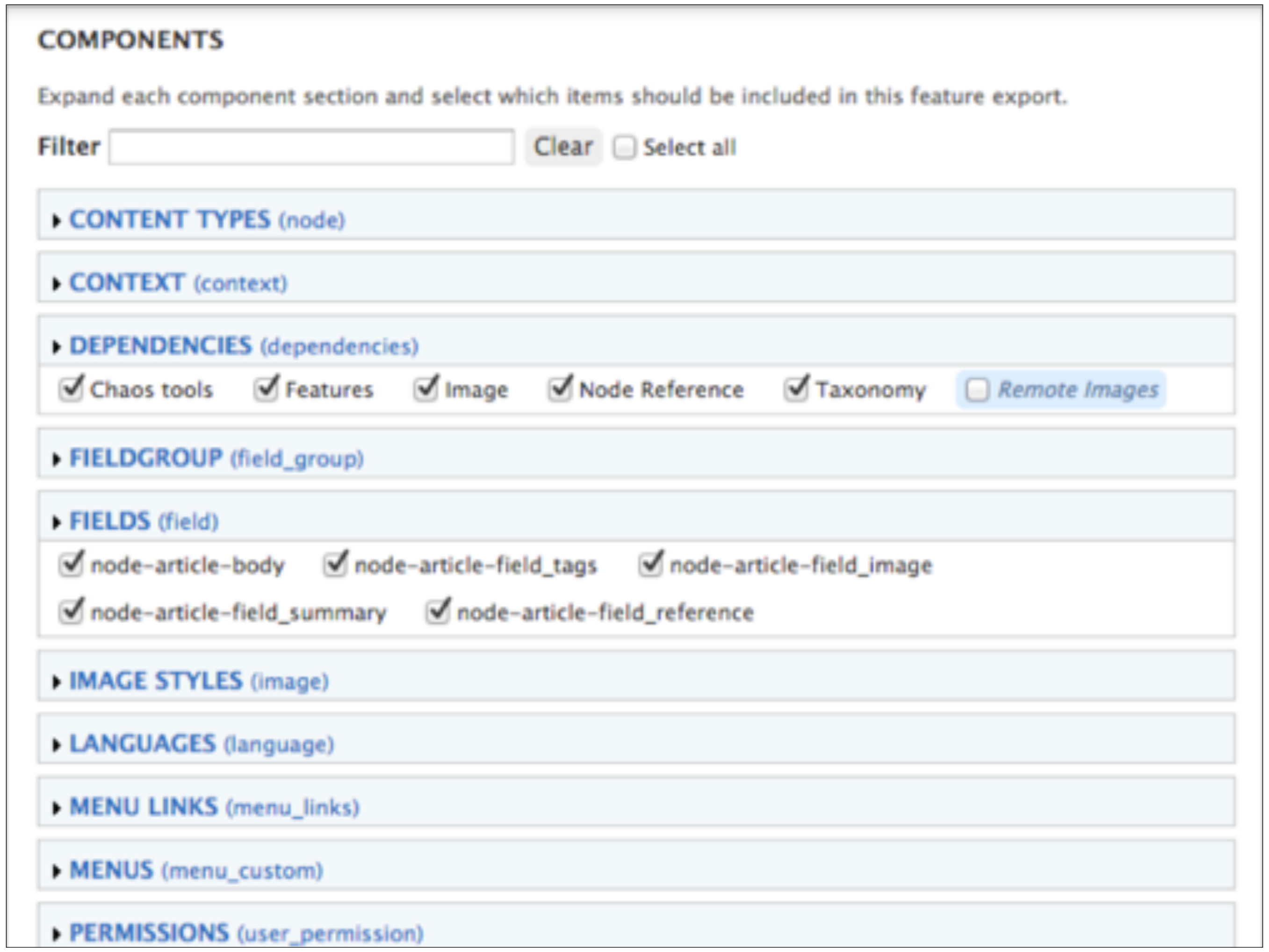
10,000ft view of how Git works with a deployment landscape (dev/stage/prod)

## Features

A feature is a collection of Drupal entities which taken together satisfy a certain use-case.

Features allow us to export point-and-clicked configuration from one environment to another.

https://drupal.org/project/features


COMPONENTS

Expand each component section and select which items should be included in this feature export.

Filter [ ] Clear ☐ Select all

▸ CONTENT TYPES (node)

▸ CONTEXT (context)

▸ DEPENDENCIES (dependencies)

✓ Chaos tools ✓ Features ✓ Image ✓ Node Reference ✓ Taxonomy ☐ *Remote Images*

▸ FIELDGROUP (field_group)

▸ FIELDS (field)

✓ node-article-body ✓ node-article-field_tags ✓ node-article-field_image

✓ node-article-field_summary ✓ node-article-field_reference

▸ IMAGE STYLES (image)

▸ LANGUAGES (language)

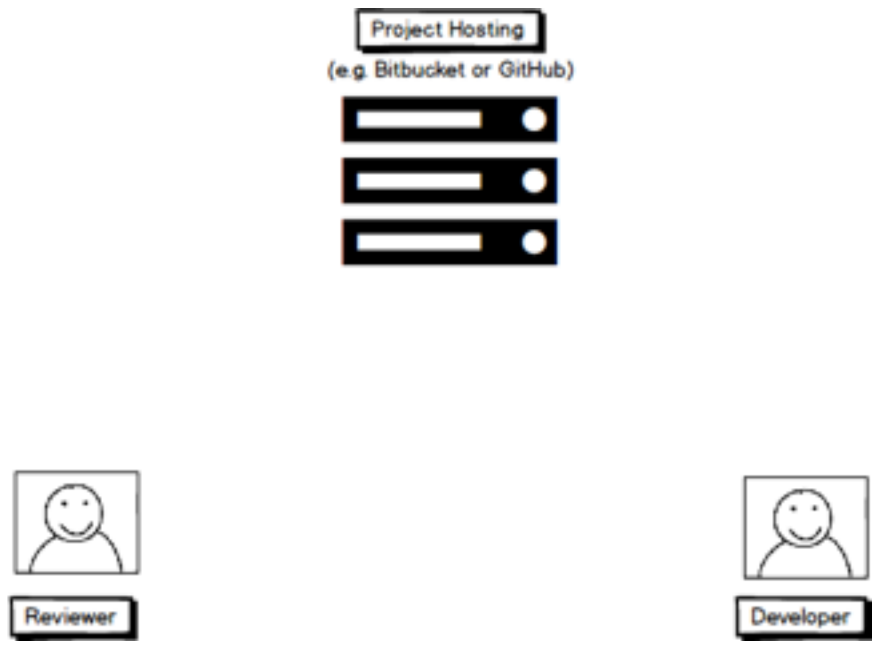
▸ MENU LINKS (menu_links)

▸ MENUS (menu_custom)

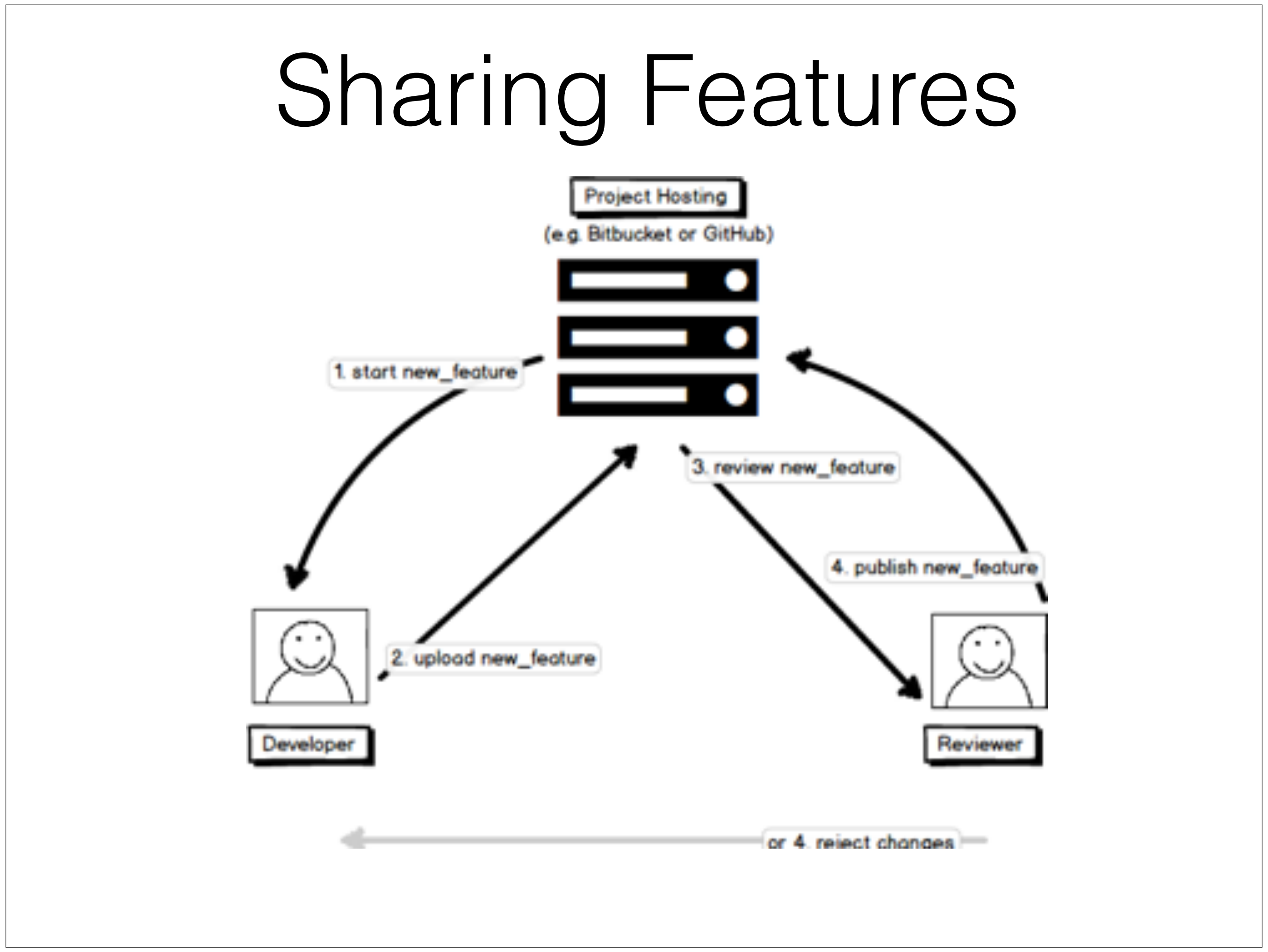
▸ PERMISSIONS (user_permission)

```
/* Sort criterion: Content: Post date */

$handler->display->display_options['sorts']
    ['created']['id'] = 'created';

$handler->display->display_options['sorts']
    ['created']['table'] = 'node';

$handler->display->display_options['sorts']
    ['created']['field'] = 'created';

$handler->display->display_options['sorts']
    ['created']['order'] = 'DESC';
```

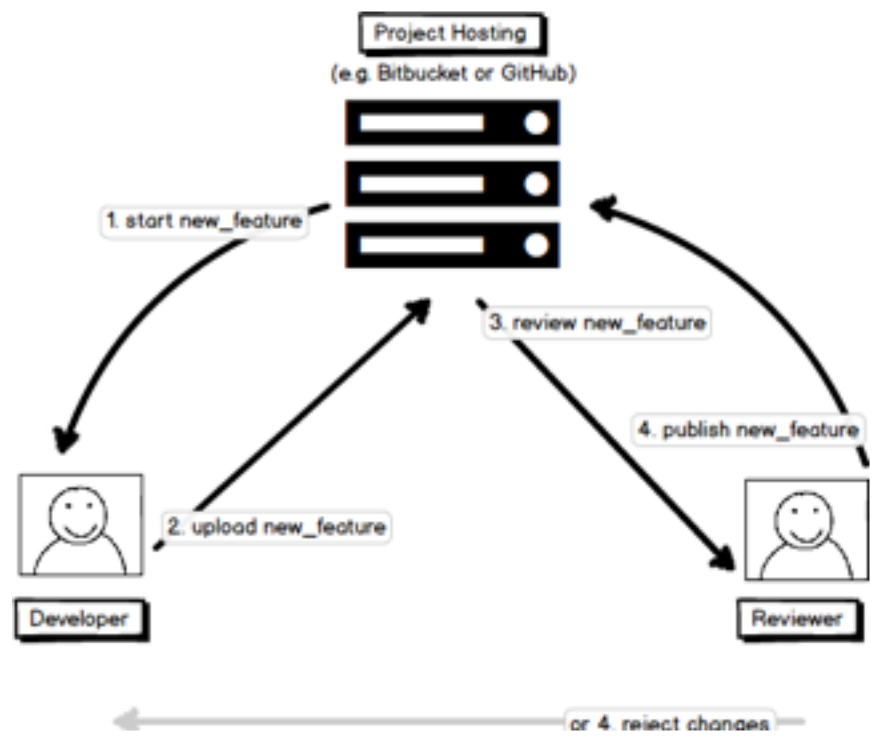
## Sharing Features



- Project hosting system, e.g. Bitbucket or GitHub or something else.
- Developers
- (who might also be) Reviewers



# Sharing Features



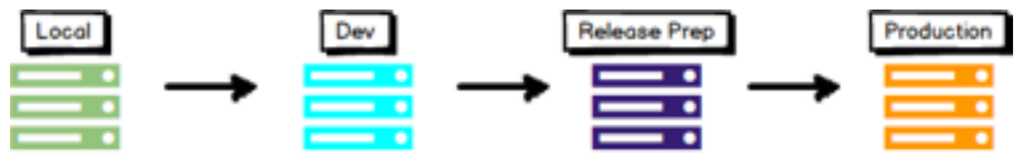
1. Start working on a feature.
2. Upload new feature.
3. Start peer review process.
4. Accept / reject the changes.

10,000ft View

## Deploying Code with Git

10,000ft view of how Git works with a deployment landscape (dev/stage/prod)

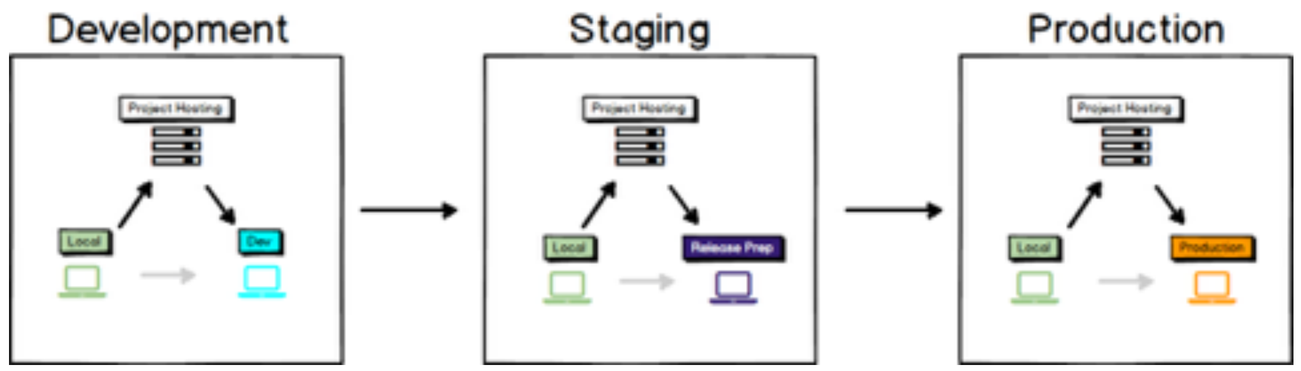
## Deploying Code



And I definitely don’t want to tell you that Git is easy to learn, because that would make me a liar.

(Actually)

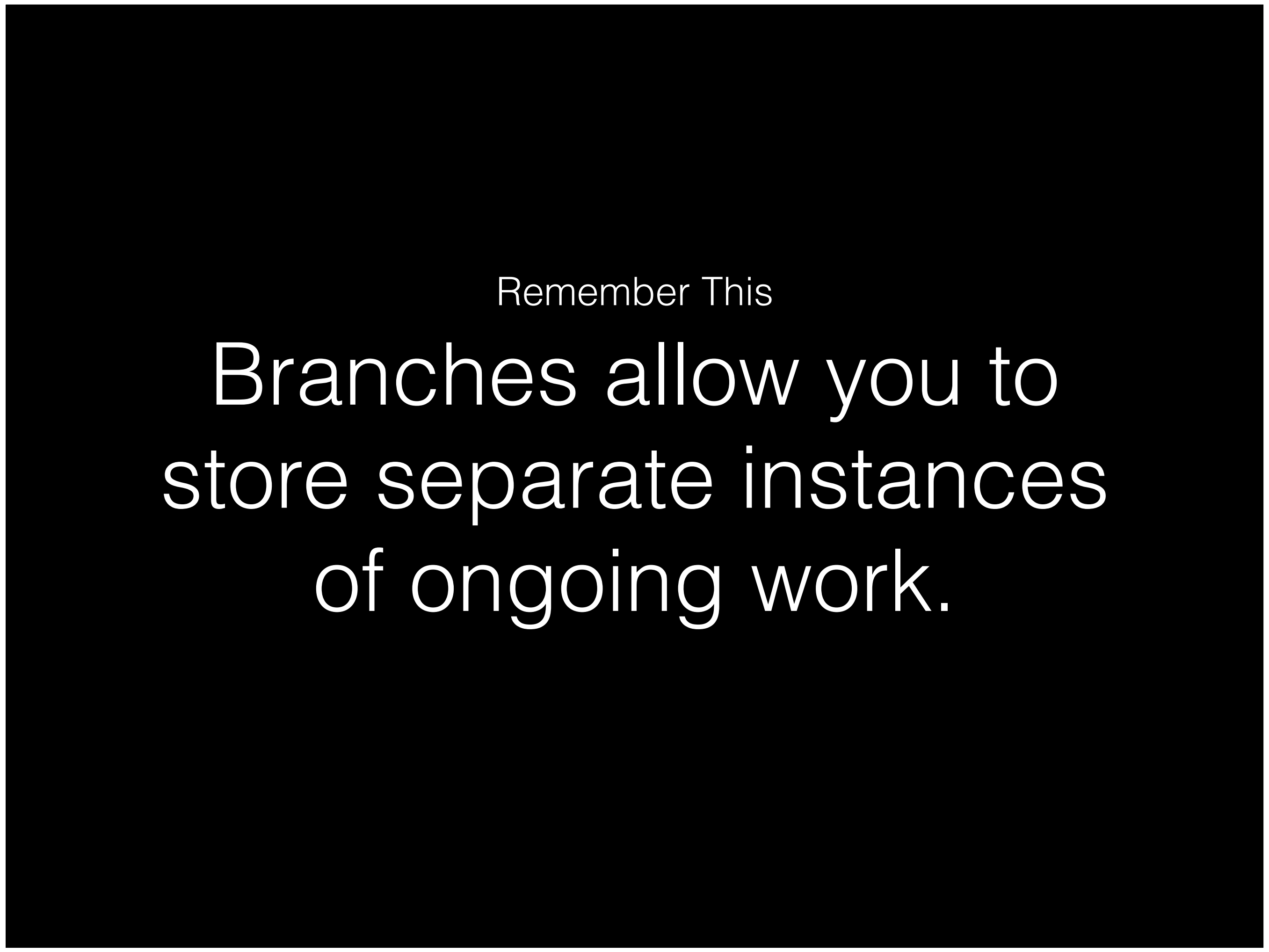
## Deploying Code



When you deploy code, you don't really move it through a series of machines.

- You push your code to the central code hosting repository
- Move to the next machine
- Pull the updated code into the new environment.

This cycle is repeated for each of the different environments that you work in (development; qa/staging; production).


Remember This

## Branches allow you to store separate instances of ongoing work.

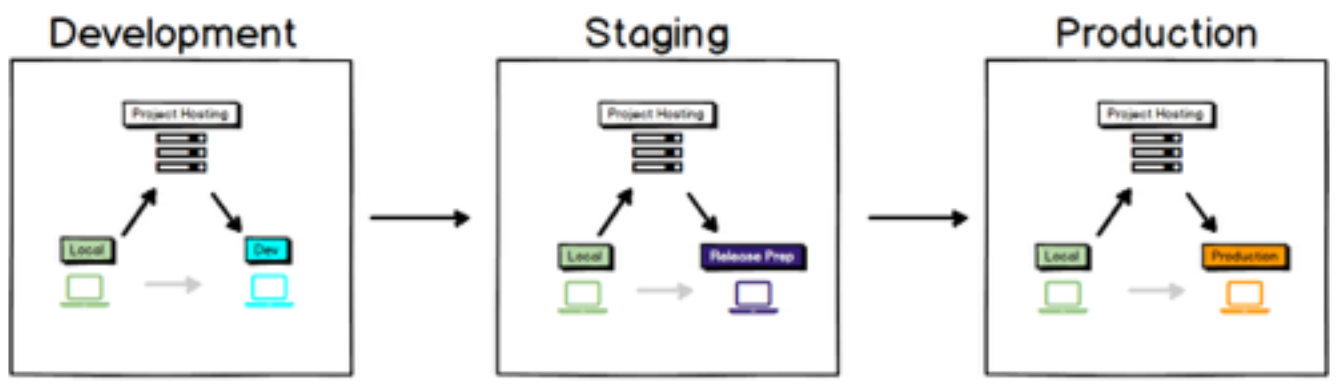
5,000ft view of how branches work, and what to do in Git world before you export a Feature

OUTCOME: Describe a basic Git branching strategy, and how it relates to a typical dev/stage/prod infrastructure

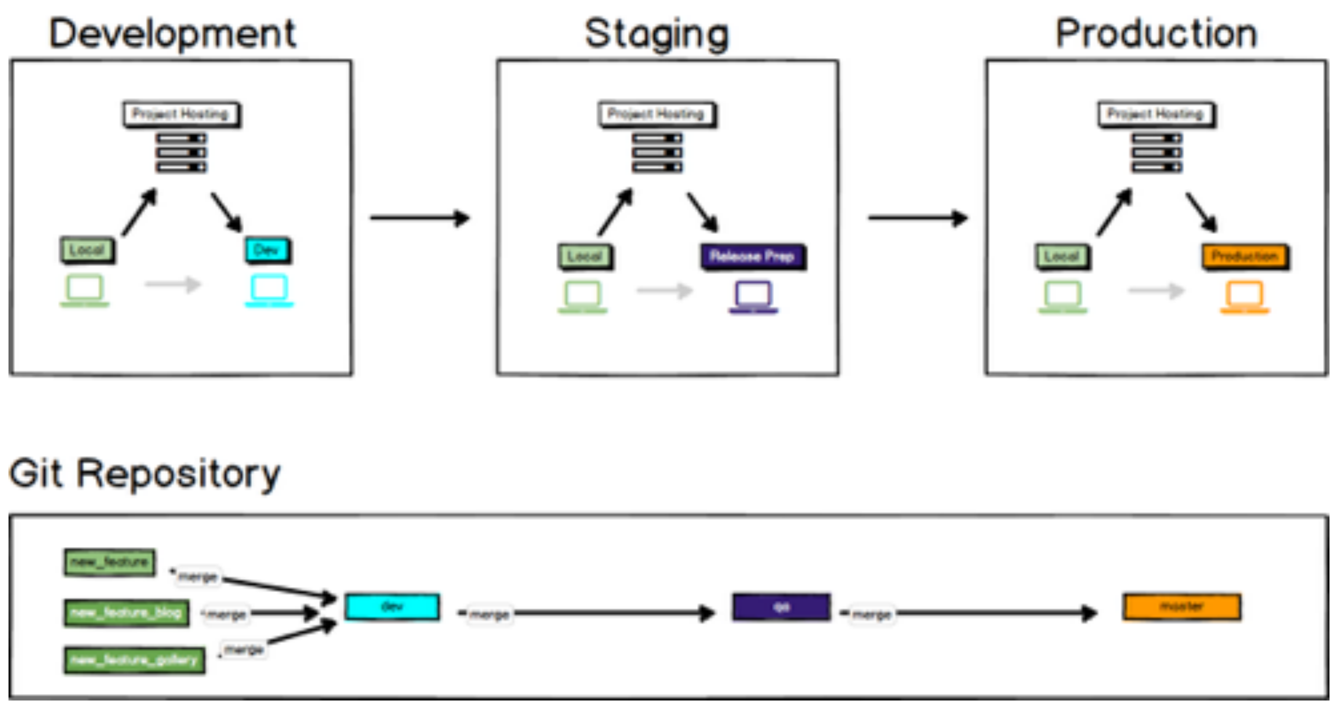
Remember This

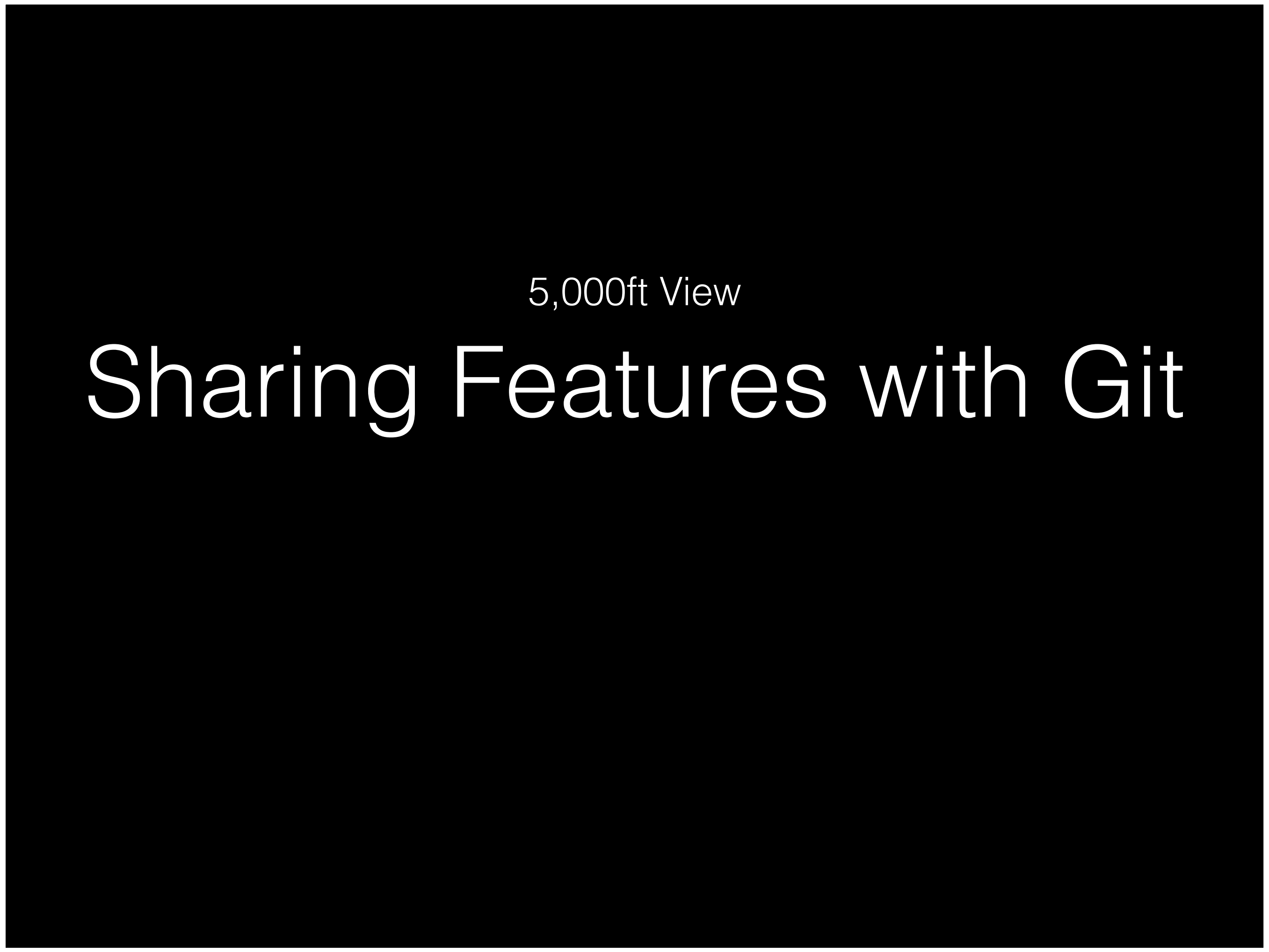
Git branching strategies are **conventions** we use based on common deployment setups.

## Per-Environment Branches



# Per-Environment Branches



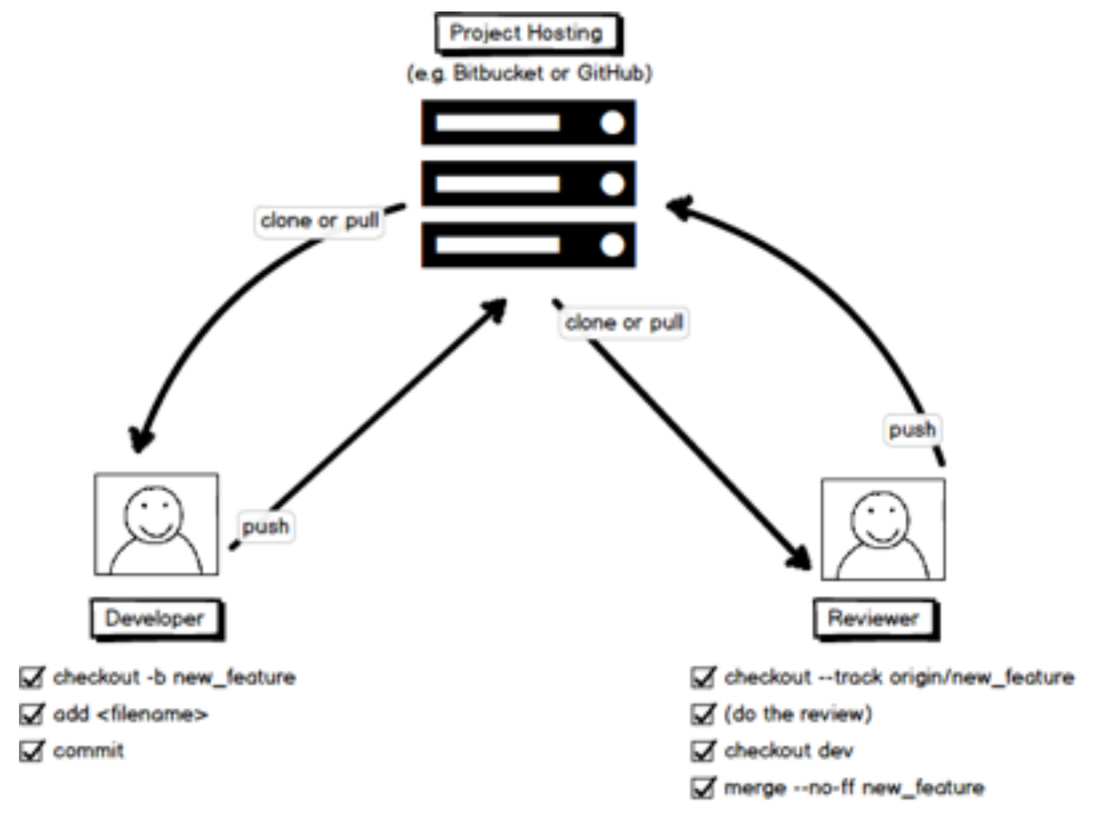


5,000ft View

## Sharing Features with Git

10,000ft view of how Git works with a deployment landscape (dev/stage/prod)

## Sharing Features with Git



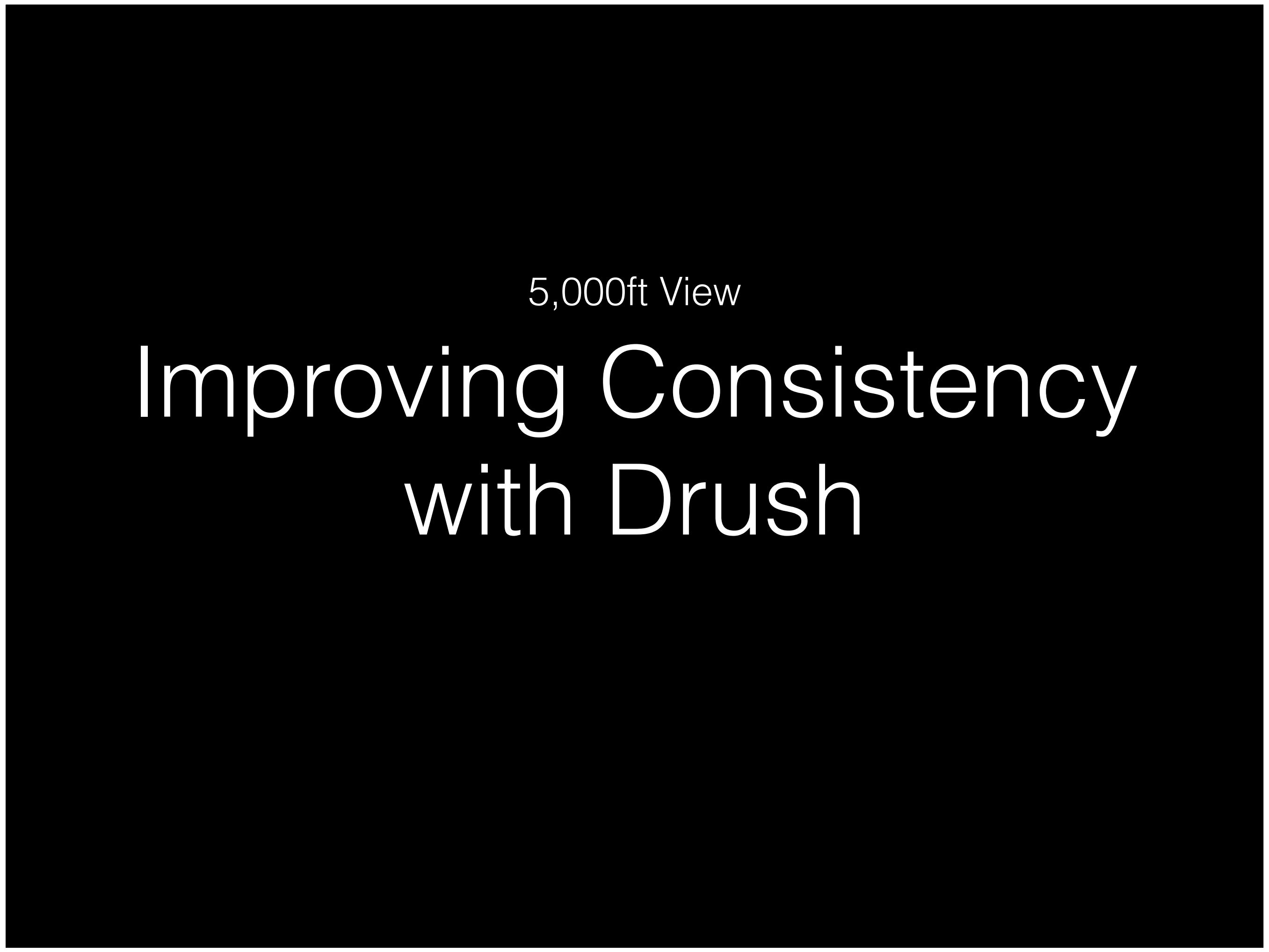
- ☑ checkout -b new_feature
- ☑ add <filename>
- ☑ commit

- ☑ checkout --track origin/new_feature
- ☑ (do the review)
- ☑ checkout dev
- ☑ merge --no-ff new_feature

- Project hosting system, e.g. Bitbucket or GitHub or something else.
- Developers
- (who might also be) Reviewers

Remember This

Features (and its related export functions)
is not always perfect but it is
always better than using nothing.



5,000ft View

## Improving Consistency with Drush

The ideal world isn't quite how Features works because the underlying modules' export tools can sometimes be inconsistent.

For example, any ctools-based module is using ctools Export function.  So if there is actually a problem with the export, it's usually the fault of the specific module.  I've seen modules that inconsistently deal with their data types (0 vs '0'), or depend upon the exact order items are added to their arrays (Panelizer).  But these days this is still pretty uncommon.

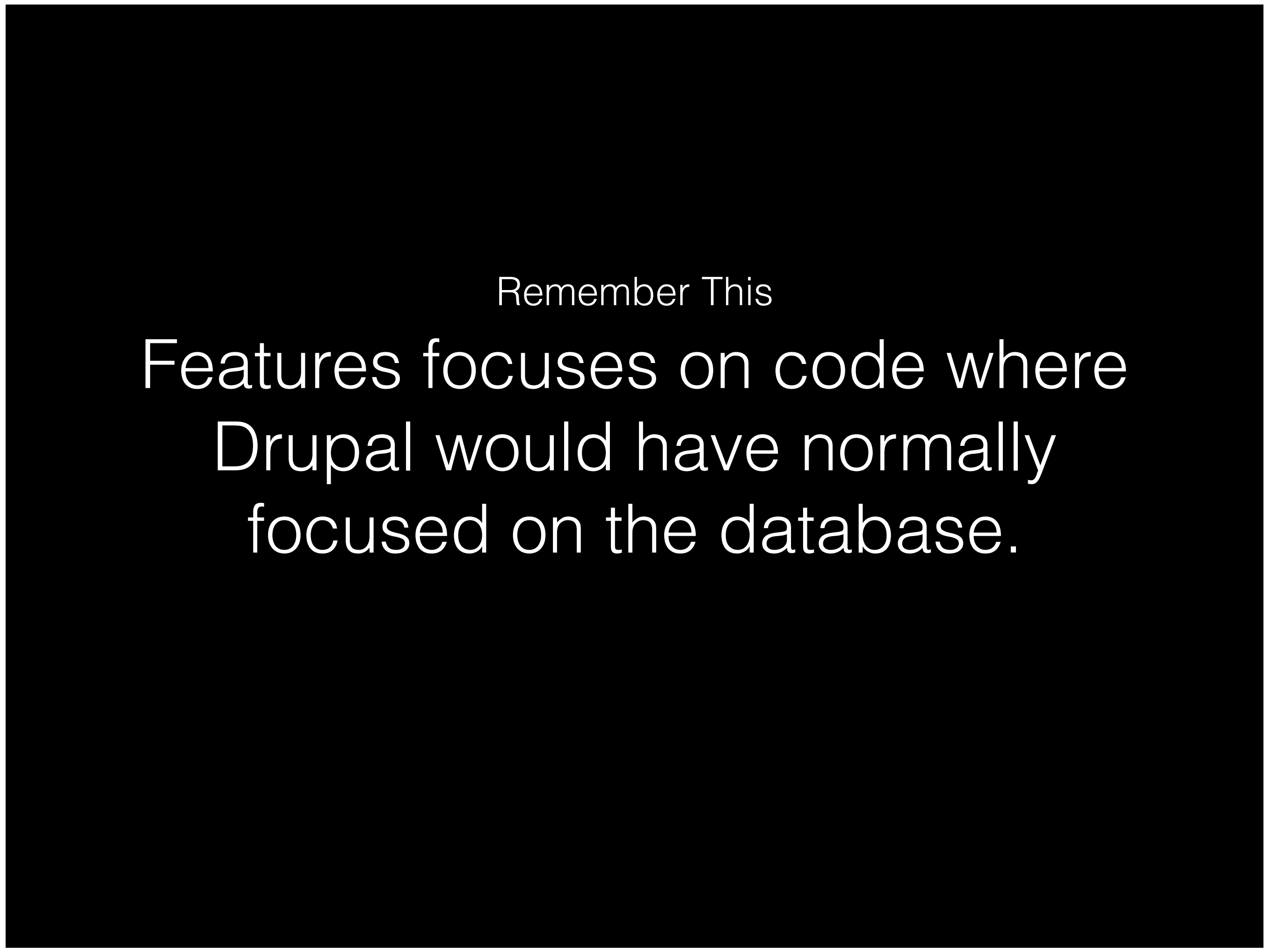
Other common problems are multiple developers **exporting Features with slightly different content config on their site**, or after installing different modules to test something that leave stuff behind in the database.  I always tell developers after doing a features-update (export) to do a git-diff to see exactly what they have really changed to make sure they don't commit/push bad code.  But it still happens all the time.  It's not Features or Git, it's just developers not paying attention to details in many cases.

Remember This

## The command line *can* provide a faster route to a more consistent experience.

## Drush

- Drush is a command-line shell and scripting interface for Drupal.
- Execute cron, run update.php, work with Features, clear cache, and more.
- https://github.com/drush-ops/drush

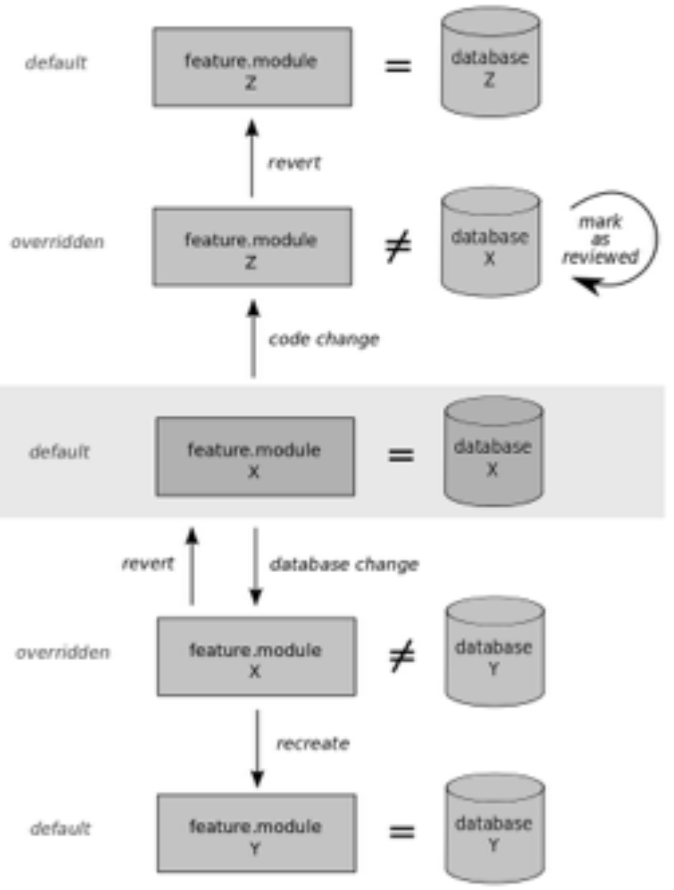


Remember This

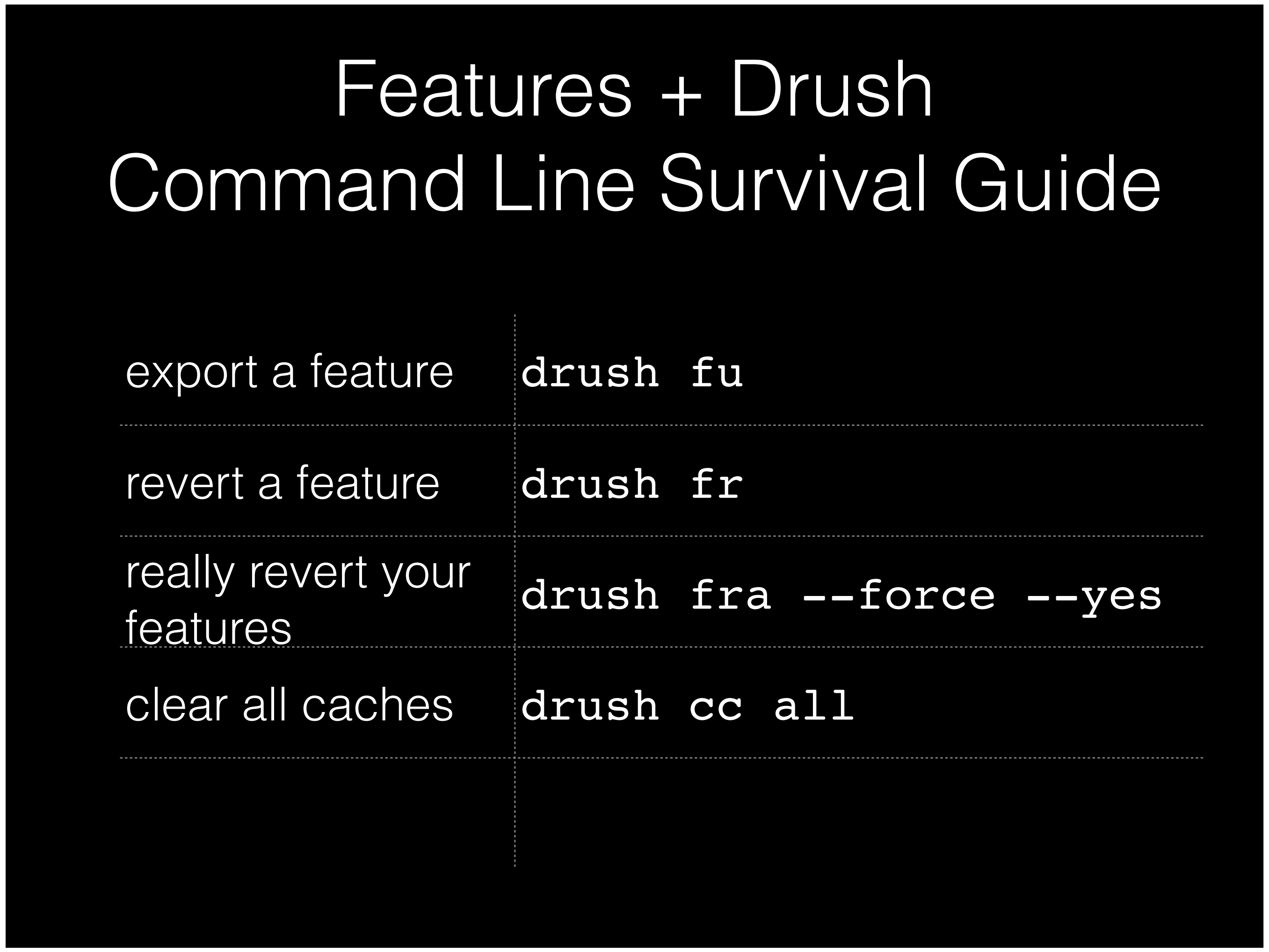
## Features focuses on code where Drupal would have normally focused on the database.

Features Revert --> use the version of the Feature which is stored in code

Features Update --> export the version of the Feature which is currently in the database to code (“update” the code version)



https://www.drupal.org/node/582680



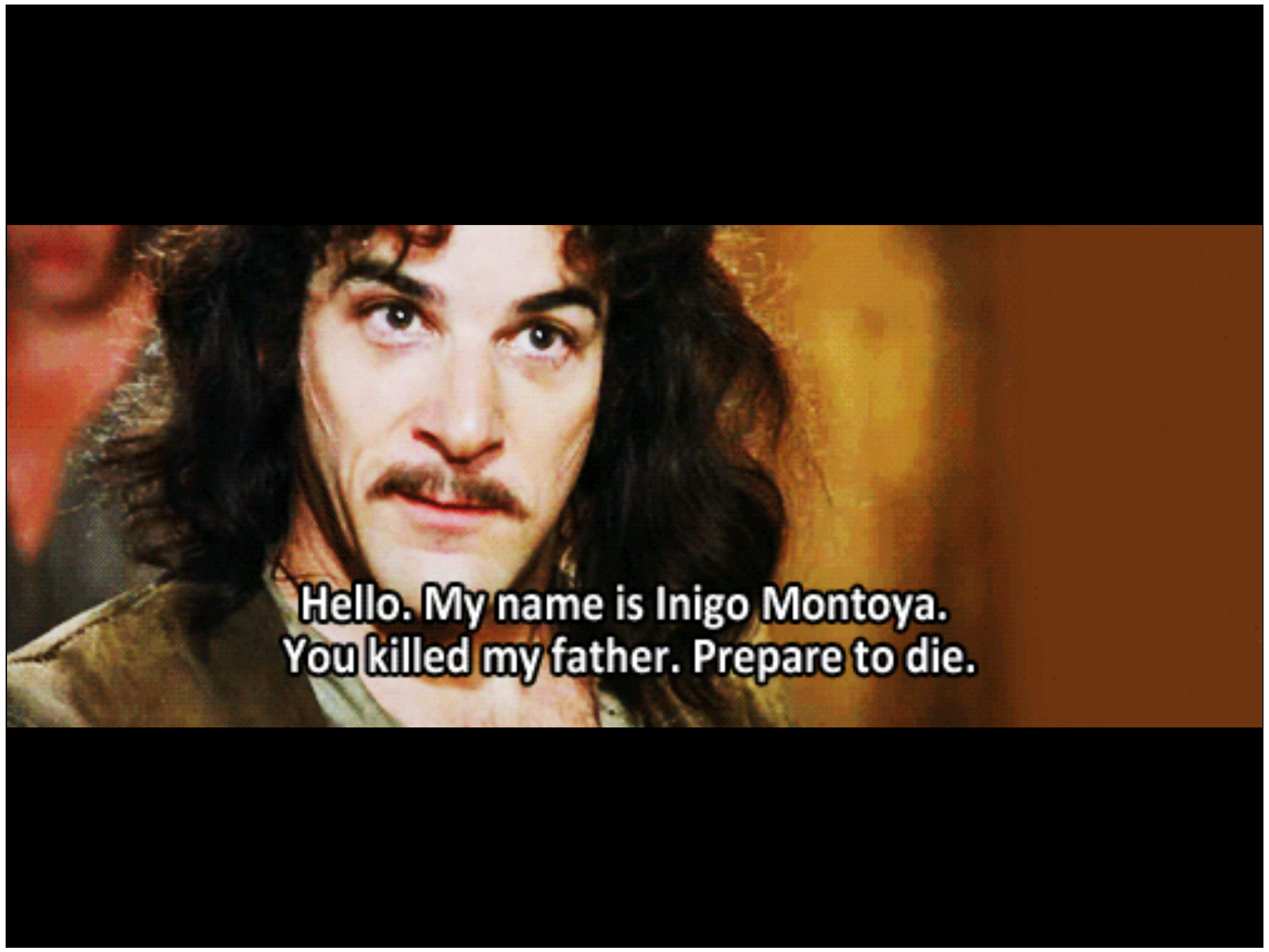
## Features + Drush
## Command Line Survival Guide

| | |
|---|---|
| export a feature | `drush fu` |
| revert a feature | `drush fr` |
| really revert your features | `drush fra --force --yes` |
| clear all caches | `drush cc all` |

https://www.drupal.org/node/582680

5,000ft View

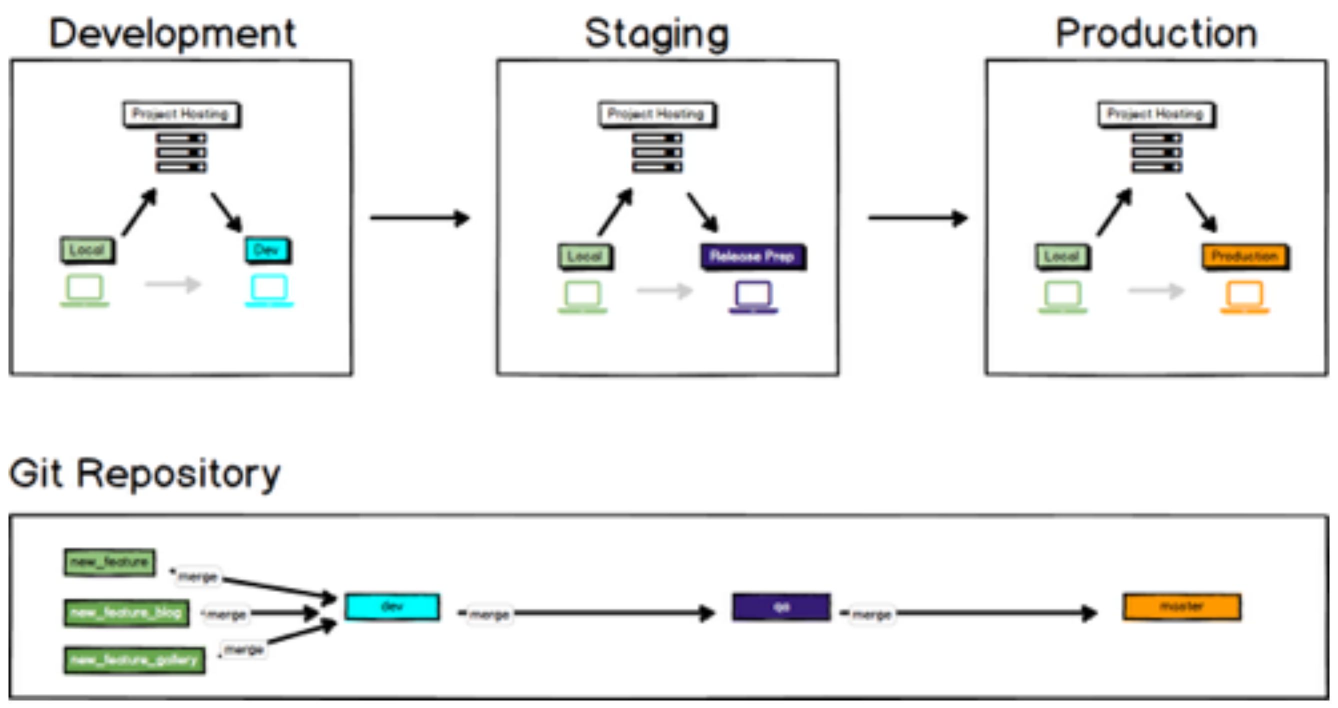
## Avoiding Conflict


Hello. My name is Inigo Montoya.
You killed my father. Prepare to die.

Remember This

## Be Unique;
## Avoid Overlap

# Branch Reminder



PREPARE YOURSELVES
WALL OF TEXT IS COMING
memegenerator.net

Sorry for the mixed metaphors.

## Getting Ready (in Git)

- Start in the right "environment"
  $ git checkout [dev]

- Create a new feature branch
  $ git checkout -b [1234-new_feature]

## Creating a Feature

(Site Builder-friendly)

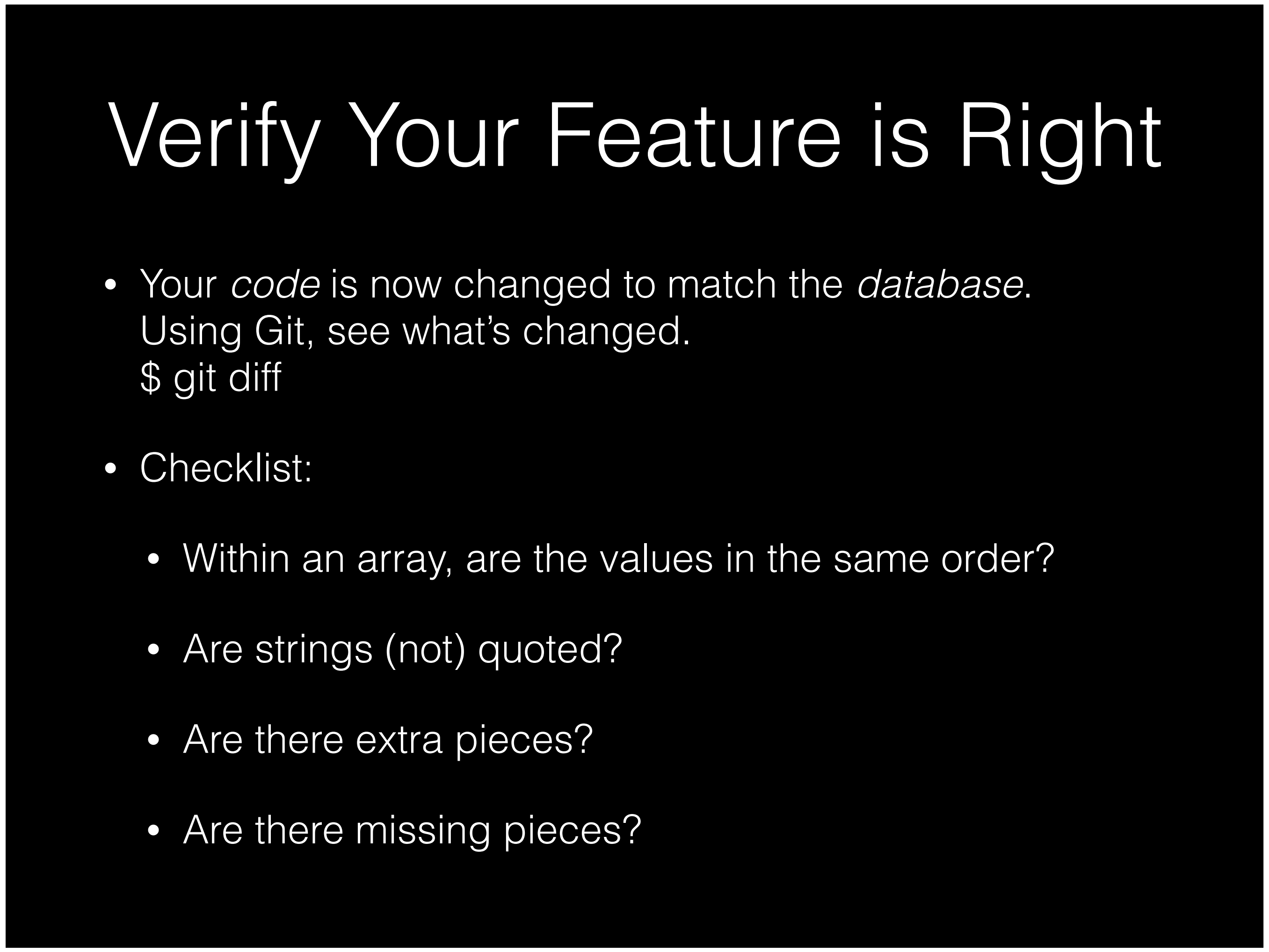
- Set all Features back to their factory defaults.
  $ drush fra --force --yes
- Build your new feature with the pointy-clicky.
- Export your feature's settings with the pointy-clicky.
- Put the downloaded file into:
  /sites/all/modules/features
- Unpack your new feature
  $ tar xvf feature_name.tar.gz

## Updating a Feature

(Site Builder-friendly)

- Set all Features back to their factory defaults.
  ```
  $ drush fra --force --yes
  ```
- Build your new feature with the pointy-clicky.
- Update all features to use settings from the database
  ```
  $ drush features-update-all
  ```
  or
  ```
  $ drush fu-all
  ```

YOU SEEM A DECENT FELLOW
I HATE TO KILL YOU



## Verify Your Feature is Right

- Your *code* is now changed to match the *database*.
  Using Git, see what's changed.
  $ git diff
- Checklist:
  - Within an array, are the values in the same order?
  - Are strings (not) quoted?
  - Are there extra pieces?
  - Are there missing pieces?

Other common problems are multiple developers exporting Features with slightly different content config on their site, or after installing different modules to test something that leave stuff behind in the database. I always tell developers after doing a features-update (export) to do a git-diff to see exactly what they have really changed to make sure they don't commit/push bad code. But it still happens all the time. It's not Features or Git, it's just developers not paying attention to details in many cases.

You don't look so good.
You don't smell so good either.

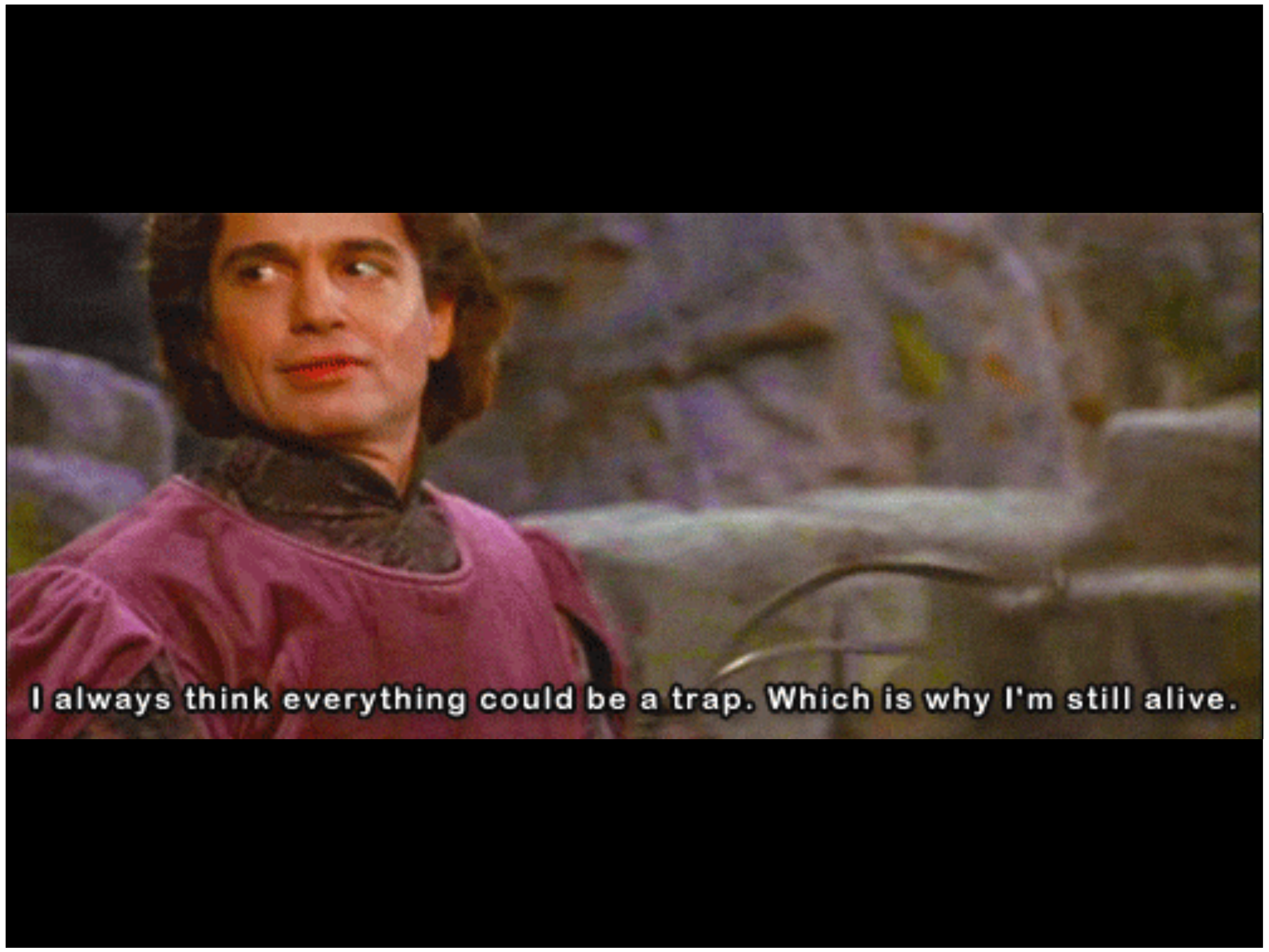
## Git It Up

- Check what is currently not in your repository
  $ git status

- Add the new Feature to Git
  $ git add [feature_directory]

- Save the new Feature to your repository
  $ git commit

- Add a really good commit message which describes what your Feature is, and all of its compoents.


I always think everything could be a trap. Which is why I'm still alive.

## Share Your Feature

- Upload the Feature to your shared code hosting repository
  ```
  $ git push origin [1234-new-feature]
  ```

## Testing Someone Else’s Feature

- Update your local list of branches
  $ git fetch

- Clean up your database by reverting all Features
  $ drush fra --force --yes

- Switch to the branch where the new Feature is
  $ git checkout --track origin/[1235-new-feature]

- Import the new Feature
  $ drush fr

## Adding a Feature to a Shared Branch

- Checkout the branch you want to add the new Feature to.
  ```
  $ git checkout [dev]
  ```

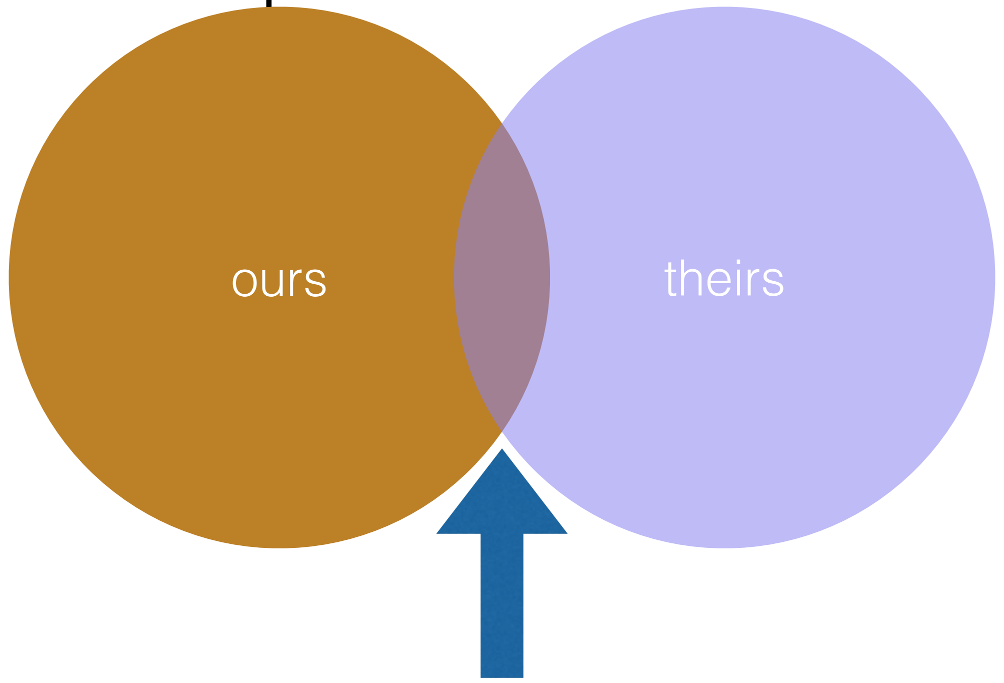
- Ensure your copy of the branch is up-to-date.
  ```
  $ git pull --rebase=preserve
  ```

- Include the new Feature into the current branch
  ```
  $ git merge --no-ff [1234-new-feature]
  ```

10,000ft View

## Dealing with Conflicts

## Conflict is when there is overlap at the same line.



Life is pain, highness; anyone who says differently is selling something

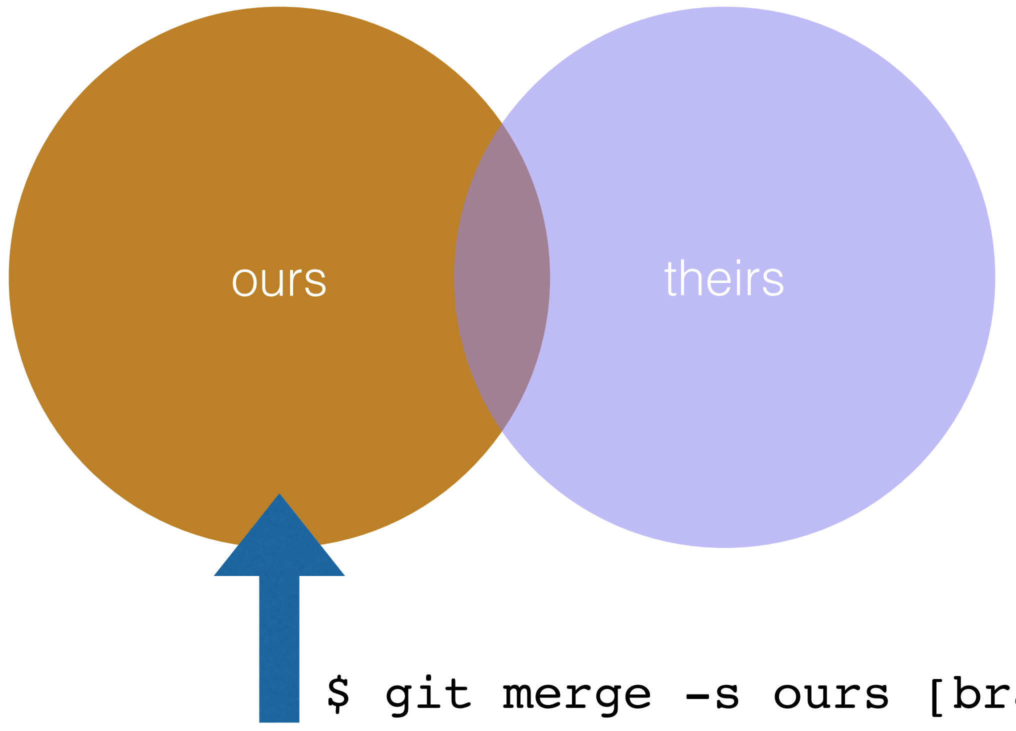
# Investigating Conflict

- Determine the point of conflict:
  `$ git mergetool`

- Want to undo that merge? Back the truck up.
  `$ git reset --merge ORIG_HEAD`

- Take another look at the differences
  `$ git diff [1234-new-feature]...[master]`

## Choose “ours”



```
$ git merge -s ours [branch]
```

## Resources

- Building a Drupal site with Git
  https://www.drupal.org/node/803746
- Git for Teams
  http://gitforteams.com

## More Resources

- Features - https://drupal.org/node/580026
- Drush - http://drush.ws/
- Introduction to Drush Series http://drupalize.me/series/introduction-drush-series
- Features & Drush Series http://drupalize.me/series/drupal-deployment-features-drush-series

DRUPALCON ★ LOS ANGELES

## WHAT DID YOU THINK?

EVAULATE THIS SESSION - LOSANGELES2015.DRUPAL.ORG/SCHEDULE

**WWW.GITFORTEAMS.COM**