Enterprise Drupal

John Jennings, Lead Developer
Johnson & Johnson Technology

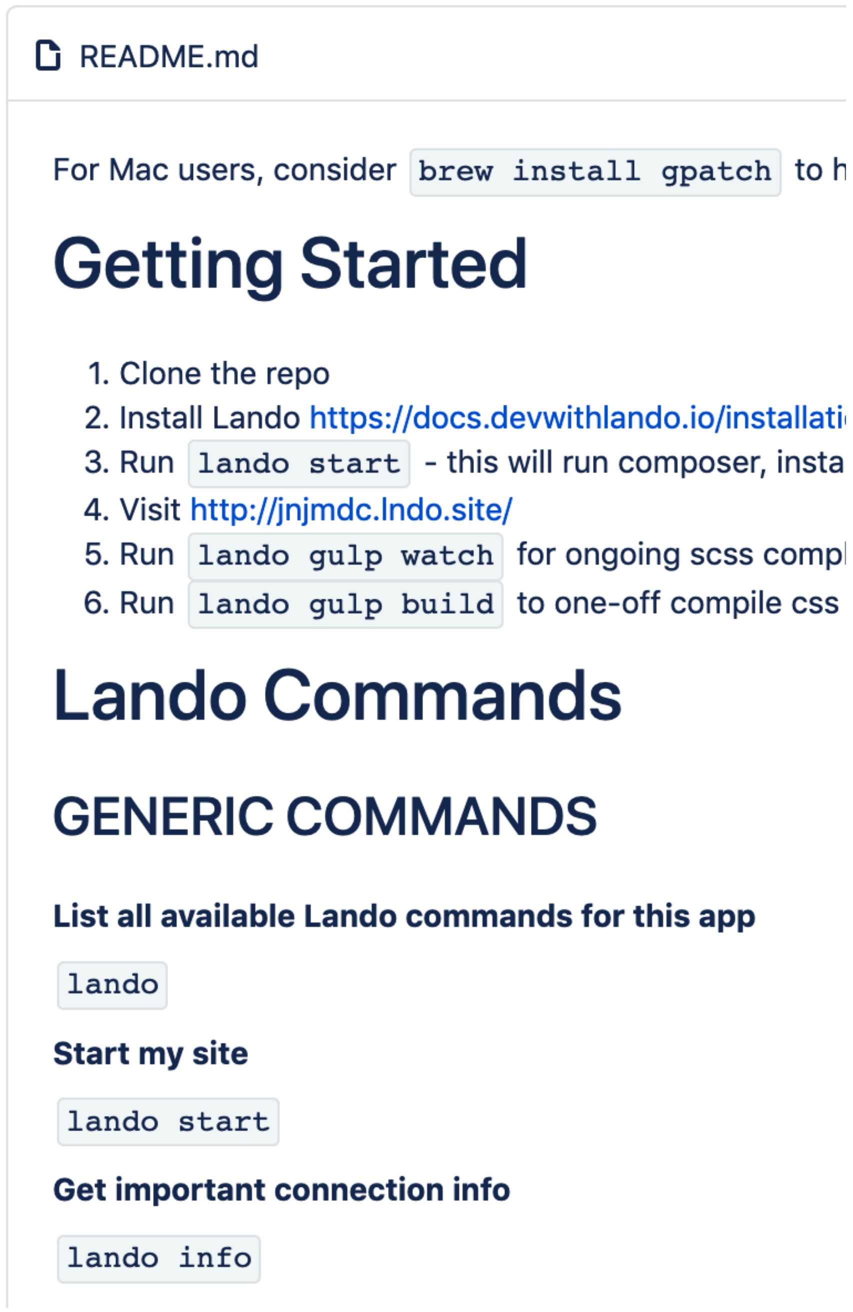
Team

- Using a large team of developers versus 1-2 leads to many differences in a Drupal project:
  - Need for communication
  - Technical alignment
  - Assignment of work
  - Remote developers

Team

- When you add a new developer(s) to the project, getting them up to speed is an essential task for other developers on the project.

- Having a good README.md file is a great way to keep track of tasks needed for initial local installation and required software.



README.md

For Mac users, consider `brew install gpatch` to h

Getting Started

1. Clone the repo
2. Install Lando https://docs.devwithlando.io/installati
3. Run `lando start` - this will run composer, insta
4. Visit http://jnjmdc.lndo.site/
5. Run `lando gulp watch` for ongoing scss compi
6. Run `lando gulp build` to one-off compile css

Lando Commands

GENERIC COMMANDS

List all available Lando commands for this app

```
lando
```

Start my site

```
lando start
```

Get important connection info

```
lando info
```

Site v. Profile

A *site* is…

- All the code necessary to make the exact website being presented when deployed.

A *profile* is…

- All the code necessary to make many different versions of a website, whether identical or sharing a common amount of code when deployed.

Site v. Profile

- A site can be deployed with configuration that is exported (via *drush cex*, for example) and imported into a production during a deployment.

- A profile (or distribution) could also contain configuration, but it would also include custom modules, features, themes, tests, custom scripts, etc. on which to base a website build or to function as a "source of truth" for a website.

Profile

- For a profile, you could use it as a starter to a website build, meaning it can be used in install/configure a website initially, but all other enhancements/updates could be controlled through normal site tasks, like a *drush config-export* and *-import*.
  - This can be referred to as an **installation profile**.

- You could also use it as the main source of code for a website, to where the website instance itself uses the code from the profile on a continuous basis, getting updates from the profile as the profile itself is updated.
  - This can be referred to as a **platform**.

Code Quality

- When you work with many developers, it is important to set standards for the work that developers commit:
  - Align the code to Drupal standards (such as using PHP_CodeSniffer's Drupal coding standards)
  - Use CSS naming conventions, like BEM
  - Use atomic design principles
  - Leverage and generalize code when possible to avoid re-writes and duplicates

Code Quality

- Developers need to give meaning to code/commits they are working on so others can follow.
- (**Note**: The presenter likes jokes! **Also note**: A pull request with a nonsensical message is good for no one!)
- Example message: “FINALLY FIXED!”
- (Good) message: “Add a translation filter to the teaser TWIG file for the news content type”

Code Quality

- When reviewing code, we use remote environments to stage changes versus the *dev* environment, and we ask the testing team to look in those environments to address changes.

- We also ask developers to not be afraid to delete or refactor code for, but with the caveat to give thorough testing, and to leverage existing code instead of accumulating debt.

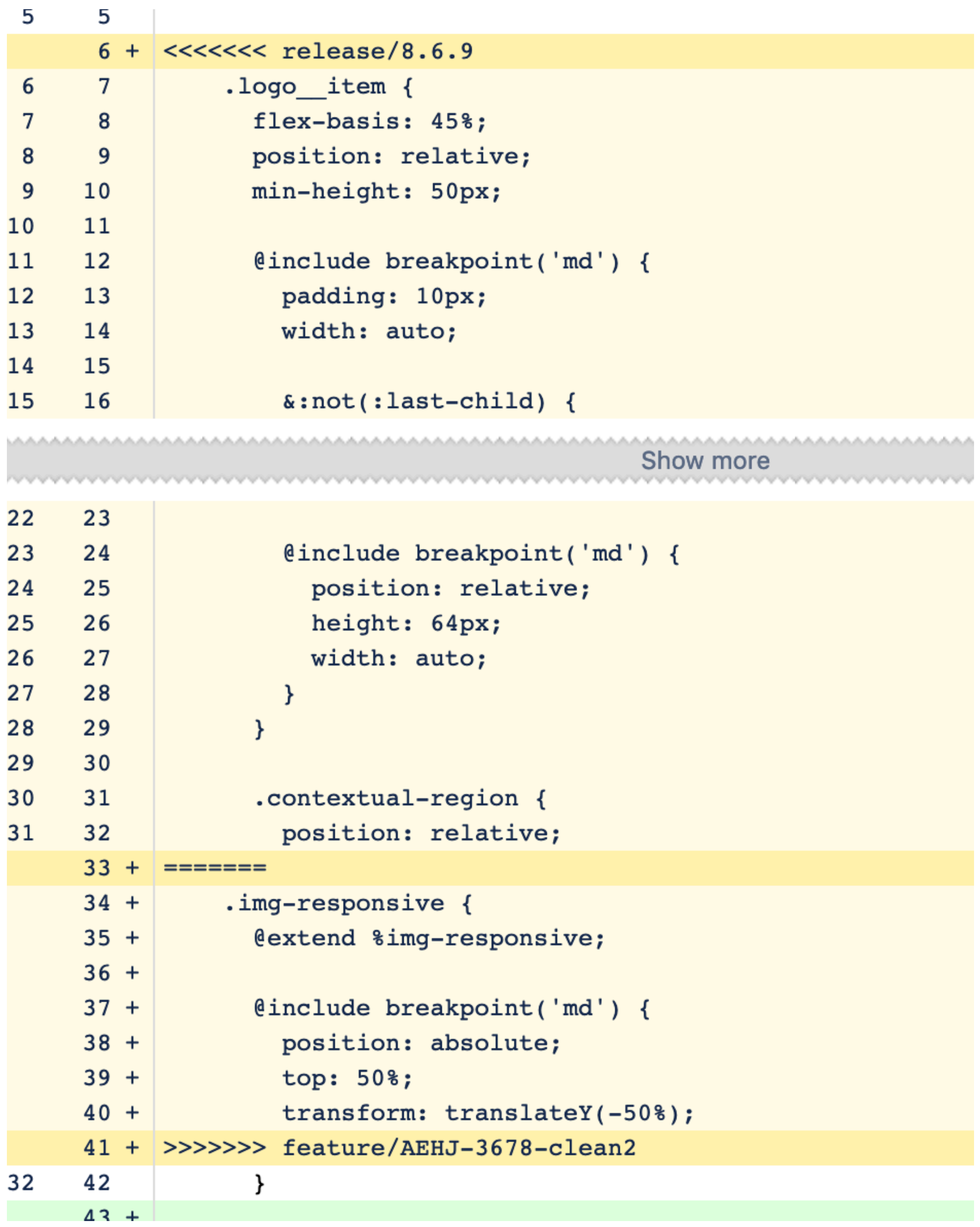
Conflict Strategies

- It is pretty common for upstream changes in the destination repository to conflict with changes in your branch, especially if you are working on the same file as the one on the destination.

- For example, two developers add new lines to the end of a file - one developer merges first to *dev* (no conflict) but the second has a merge conflict while making a pull request to *dev*.

Conflict Strategies

- For another example, the same block of code was edited by two developers, but the pull request by the second will show a merge conflict to *dev*.



```
 
+<<<<<<< release/8.6.9
     .logo__item {
       flex-basis: 45%;
       position: relative;
       min-height: 50px;
 
       @include breakpoint('md') {
         padding: 10px;
         width: auto;
 
         &:not(:last-child) {
```

```
 
         @include breakpoint('md') {
           position: relative;
           height: 64px;
           width: auto;
         }
       }
 
       .contextual-region {
         position: relative;
+=======
+    .img-responsive {
+      @extend %img-responsive;
+
+      @include breakpoint('md') {
+        position: absolute;
+        top: 50%;
+        transform: translateY(-50%);
+>>>>>>> feature/AEHJ-3678-clean2
       }
+
```

Conflict Strategies

- Two distinct ways to address merge conflicts are through a merge (from the destination) or a rebase (also from the destination).

- In a merge, you would merge the latest version of the destination branch, such as dev, into your local source branch.
    - Command: *git merge dev*

- In a rebase, you would also work with the latest version of the destination branch.
    - Command: *git rebase dev*

Conflict Strategies

- *Thinks out loud … those seem pretty similar*
- The main difference is a *merge* applies the **code/history from the destination after** your work is done on your branch while a *rebase* rewrites the **code/history of your local commit after** the commits in the destination.
- Which is better? You can decide but be consistent!

Internal Decisions

- A big factor in the success of a large development team is how to implement business requests and address fixes:
  - Having a “technical alignment” document/meeting/story can help eliminate technical debt and re-work
  - For example, decking whether to use custom code v. a community module
- Selecting how/when to adopt a community module can have many factors:
  - How widely is the module used/adopted
  - Is the module actively maintained
  - Are technical issues being addressed

Internal Decisions

- How to organize code:
  - Configuration v. features
  - Theme patterns v. one-off code
  - Templates v. view modes v. preprocessing

Communication

Do it!

Consistency

- Making good internal decisions and having a lot of communication between developers (and other stakeholders) can only go so far if your process is constantly changing.
- A development team is not going to reach their full potential (improving velocity of work completed and reducing bugs in said work) unless you have a consistent and well-understood process.

Thank You!

Questions?