Introduction and Existing Infrastructure

This tutorial is currently structured with the assumption that there are two main deployment places within the workflow; dev and prod.

Upon the creation of a staging environment, this workflow should be modified to support the addition.

The existing infrastructure includes:

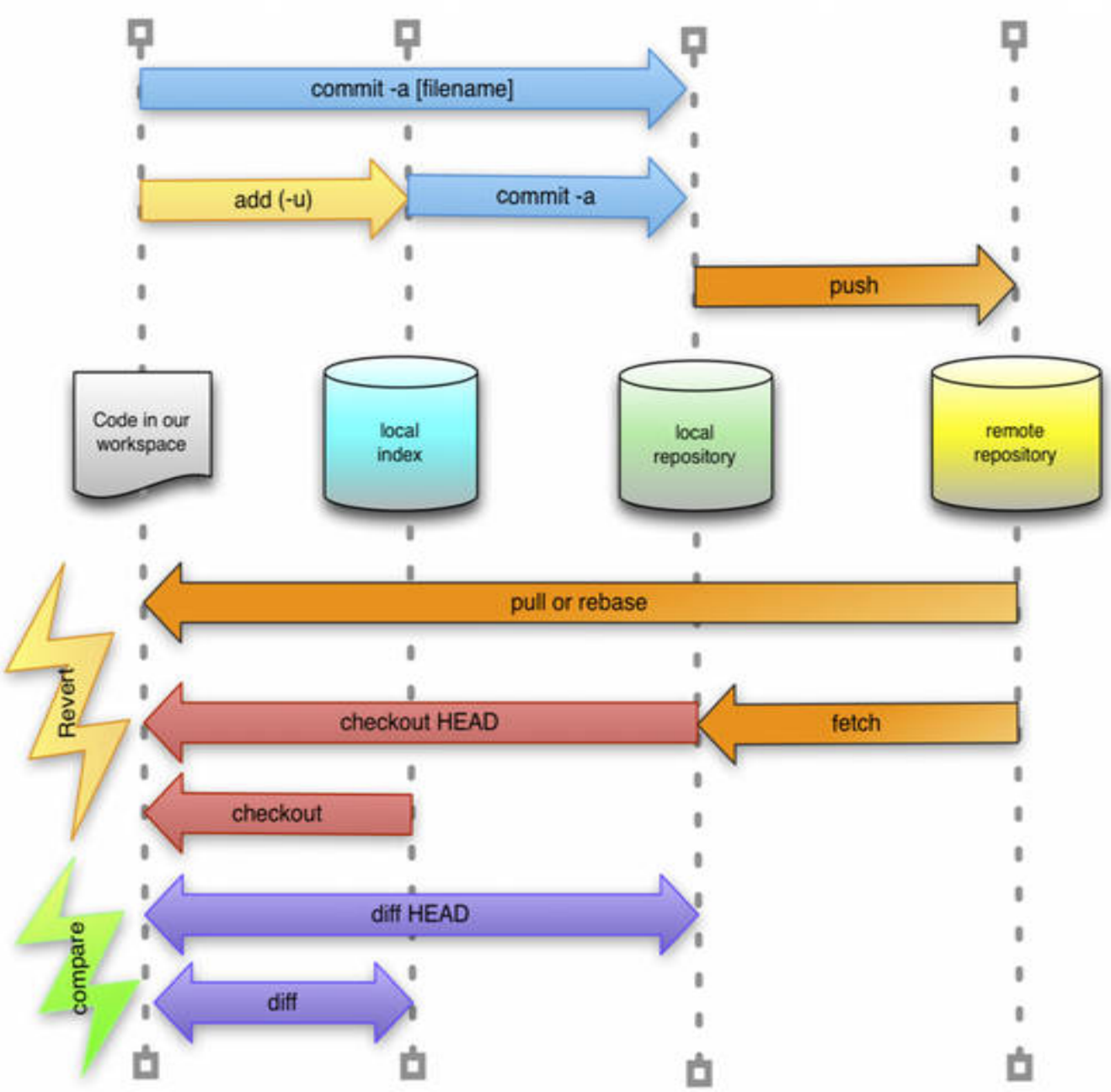
1. 1 Server that facilitates development (dev)
2. 1 Server that facilitates production
3. The division of these systems by a robust firewall.

There should be a single repository that represents the asc.gov and hotline website collectively. This is because the main repository is designed as a Drupal multi-site.

The repo will need to be instantiated in such a way that it has the ability to communicate effectively with both environments.

Repo Overview

The remote repository will either be stored directly on the server or in a remote third party location. Given bitbucket as the selected tool, this will be the central mechanism to coordinate the repository and the supporting pipeline.



Branch Creation

There should be two primary branches that the servers use in a dedicated fashion

```
git checkout -b <branch-name>
```

Commit to Repo

Each time the functionality has a major addition made that is stable, the branch should be updated and the functionality should be committed to the branch. This is true even if the functionality is not completed fully. This allows other engineers to pick up on a branch should the need arise to move the branch to another developer mid-development (though this should be a rare occurrence).

```
git checkout -b your-new-branch
git add <files>
git commit -m <message>
```

Merge to dev, Pull Request and Engineering Review

The engineer responsible for the branch should merge code to the dev branch, and then verify that the functionality works as designed on the dev server.

In some cases, it may be desirable for the customer, project manager, or other team members to review the functionality. Given the lack of a staging environment, this may involve manually walking the customer through the functionality on dev.

Upon approval of the functionality on dev by the engineer responsible for the branch, a pull request should be created. This will initiate the engineering review by a lead developer. If changes are requested to the PR, comments should be made on the PR and assigned back to the developer to apply those changes.

Once the PR passes engineering review, it is ready to merge to main.

Merging to Main

Once the pull request has cleared engineering review and an approval is given, it can be merged into the main branch and deployed upstream.

Upon merging, bitbucket should automatically deploy the repo to the prod server in an ideal setup.

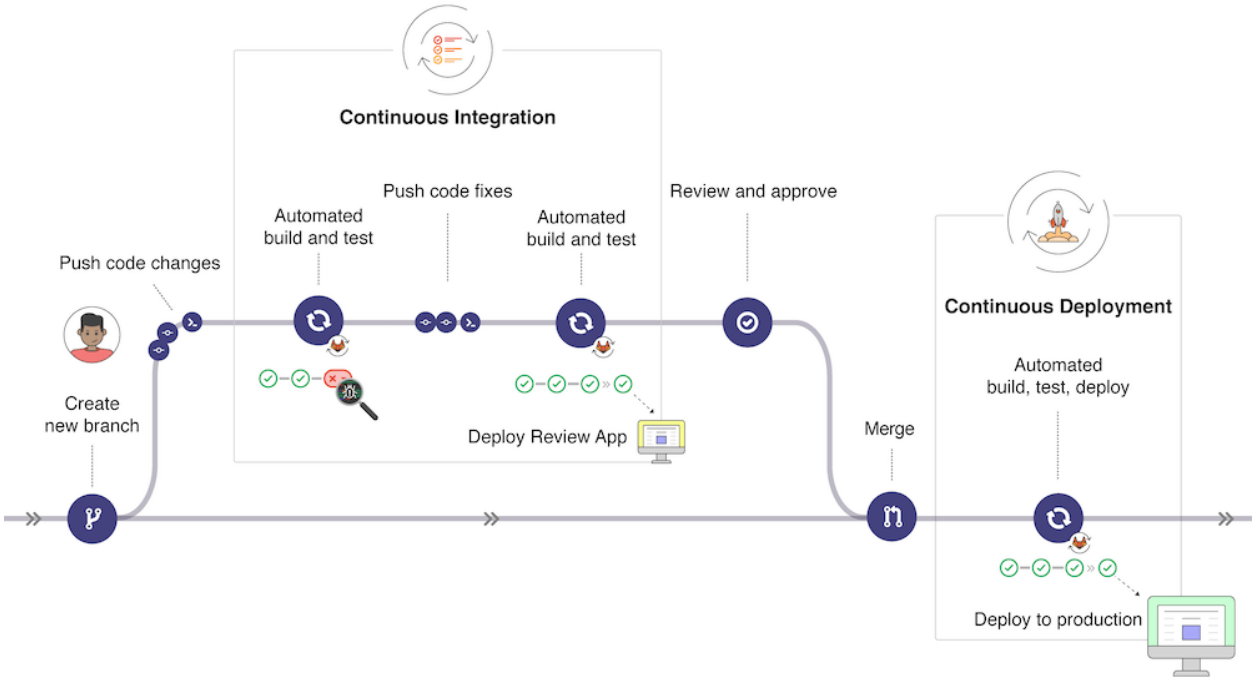
Once verified on dev, the repo's main branch should have the new code branch merged in, and then main should be deployed to production.

While ideally, old branches should be deleted upon merging, it may be wise upon initial inception that we keep old branches even after merging due to the lack of supporting infrastructure and newness of the git process flow on the system as a whole.

Other Tools

Git Desktop is a great tool for managing branches locally using a UI. This free tool can be downloaded here:
https://desktop.github.com/

Repository Pipelining



Once the repo is created, there should be pipeline controls that act as continuous integration support. This will put certain checks and balances in place to lessen the chance that code committed will break existing infrastructure as a result of preventable issues, such as code that has syntax errors, malformed code, etc.

This should be implemented on both the dev and production branches (dev and main), though the nuances may be slightly different on those controls.