WebDev Git Workflow

November 2017

Overview

Releases

- Weekly
    - 90+% of our work
- Fast Track
    - Exceptions

Branches

- Master
    - Production = inmotionhosting.com
- Staging
    - Staging = staging.dev.imhwebdev.net
- Release
    - Staging = same environment as above
- Ticket(s)
    - Dev Environments = username.dev.imhwebdev.net

Weekly Workflow

- New ticket branch created for each ticket
- When finished, ticket branch is merged into staging for review
- Once approved, ticket branch is merged into master on specified day

Fast Track Workflow

- Same as above, except

    - Merge into staging may be skipped
    - Ticket branch merged into master any day of the week
- Important things to note
    - No commits directly to master, always create a ticket branch
    - All fast tracks should be approved by manager

Rationale

Summary

- You may have read the above and thought ‘what’s the difference between that and what Ken’s team is doing? Is ‘staging’ just ‘dev’ with another name to confuse us? Why does Jimi always torture us with things to read?’ If you’re interested in the differences, read on. If not, skip this part and go to the next section

References

- Dev Git Workflow
    - Ken’s team essentially uses Gitflow
    - Gitflow works well for a team preparing code for a bundled release, where the majority of the code is released at the end of the sprint
    - In contrast, our work is often split between ‘ready for release’ and ‘under review’, frequently mixing small tasks such as text revisions with projects that span multiple sprints
- Feature Branch Workflow
    - On the plus side, the need to release small changes quickly suggests a more simplified approach, similar to the Feature Branch Workflow, where feature branches equal ticket branches
    - However, in addition to ticket branches, we also need a shared branch for review, hence the staging branch, which corresponds to the staging environment
- Similar Ticket / Review Workflow
    - The following variation is very similar to our situation and needs

    - “We used a Scrum Agile methodology with well defined sprints of work and deliverables at the end of each sprint”
    - “We released once a week, and only those changes that have been approved by the business owners in QA got merged into master and released”
    - “Everything branches off master. Features and hotfixes are treated the same”

Git Commands

Create New Branch

- Create new ticket branch
    - git fetch origin
    - git checkout master
    - git pull origin master
    - git checkout -b IMH-99
    - git push -u origin IMH-99

Commit Changes

- Commit changes to the ticket branch
    - git commit -am ‘message’
    - git push

Stay Up To Date

- Merge master into your ticket branch frequently to stay current
    - git checkout IMH-99
    - git pull origin master
    - git push

Merge Into Staging

- When ready for review, merge ticket branch into staging
    - git fetch origin

    - git checkout staging
    - git pull origin staging
    - git merge --no-ff IMH-99
    - git push origin staging
- Update staging (SSH into staging)
    - git fetch origin
    - git checkout staging
    - git pull origin staging
    - [github username & password]

Databases

Summary

- You will first need to set up the database for your environment, using the steps at the Database Setup section of the IMHWebDev.net doc

Create SQL Copy

- The first step will be to create a SQL file that is a copy of either the staging or production database. The commands below will add the file you create to the /databasedump directory
- To copy staging, SSH into staging, then run
    - mysqldump -p -u staging_example --databases staging_example > ~/databasedump/promo-2-2.sql
- To copy production, SSH into production, then run
    - mysqldump -p -u prod_example --databases prod_example > ~/databasedump/promo-2-2.sql

Transfer SQL Copy

- Through SFTP, download the SQL file you created above (promo-2.2.sql or something similar) and then upload that same file to the /databasedump directory in the environment you'd like to update (your dev account, staging, or production)

Vim Summary

- vim databasedump/promo-2-2.sql
- i (= insert)
- Edit (comment out lines)
- Escape key to exit insert mode
- :wq (= write / quit)

Update Your DB

- Once you have edited the sql file successfully, use the following command, substituting your username, database, specific sql filename, etc.
    - mysql -p -u username_imh --database username_example < ~/databasedump/promo-2-2.sql
- Enter the DB password for your DB user

Update Staging DB

- To update staging, SSH into staging, then run
    - mysql -p -u staging_imh --database staging_example < ~/databasedump/promo-2-2.sql

Configure Test.php

- To see the database changes on the front end, you will need to link your databases to your account
- Edit the following file: resources/config/test.php
- Update lines 10-12 and 17-19 so that both arrays have essentially the same info
    - dbname => ‘username_example’
    - user => ‘userame_imh’
    - password => ‘your database user password’
- It’s important to note that you should not commit changes to this file, as it includes your individual password to your dev environment
- When you’re done testing, you can simply checkout the remote copy of this file to overwrite your changes (git checkout resources/config/test.php)

Release

Pull Origin

- Always start the release process by syncing your local with origin master
    - git fetch --all --prune
    - git checkout master
    - git pull origin master

Tag Previous Release

- Tag the previous release prior to creating the new release branch
    - git tag -a 2018.1.17 -m "Yoda mid/fix/end" master
    - git push --tags

Create Release Branch

- Pull Origin first, make sure you're up to date with origin master
- Create new ticket branch
    - git checkout -b yoda-mid
    - git push -u origin yoda-mid

Merge Ticket Branches

- WebDev Releases should have a list of ticket branches ready for release
- Merge all finished ticket branches into release branch
    - git merge --no-ff origin/IMH-99
        - [repeat above command x10, x20, etc.]
        - [if merge conflict, see merge conflicts]
    - git push origin yoda-mid

Update Staging

- SSH into staging

    - cd inmotionhosting.com
    - git fetch --all --prune
    - git checkout yoda-mid
    - git pull origin yoda-mid
    - [github username & password]

Merge Into Master

- Merge approved tickets into master
    - git fetch --all --prune
    - git checkout master
    - git pull origin master
    - git merge --no-ff origin/yoda-mid
    - git push origin master

Push Live

The Magic Button

- Since the production site is still connected to the virtual boxes, we still need them to push our code live. So, ‘pushing the magic button’ means loading up **your virtual box** and running the following
    - git fetch origin
    - git checkout master
    - git pull origin master
    - git push live master

Conflicts

Merge Conflicts

- Depending on how much code you try to squash together all at once, you will inevitably run into merge conflicts. The message will be something along the lines of ‘Unmerged paths: both modified /styles.css’
- To resolve
    - [Edit the file]
    - git commit -am ‘resolved merge conflict’
    - git push origin
- To abort
    - git merge --abort

Reverts

- Reverting an individual commit is pretty straightforward
    - git revert b8bd171...
- Reverting a merge commit requires a bit more (see: https://git-scm.com/blog/2010/03/02/undoing-merges.html)
    - git revert -m 1 b8bd171...

Cleanup

Prune The Tree

- Prune your local git tree
    - git remote update origin --prune

Create New Staging

- Fetch and prune
    - git fetch --all --prune
- Delete previous staging
    - git branch -D staging
    - git push origin --delete staging
- Create new staging
    - git checkout -b staging
    - git push -u origin staging

Staging Reset

- Delete local staging
    - git branch -D staging
- Fetch and prune
    - git fetch --all --prune
- Checkout new staging
    - git checkout staging
    - git pull origin staging