git-flow

...

sync-up

Overview

- Release 1713
- Process Review
- Tools
- Going Forward
- TODO

Release 1713

- First release that (mostly) followed process
- rel1713 (release integration) branches created from appropriate hashes in
  - 51 FOSS repos
  - 48 NETWORK repos
  - zm-build, zm-network-build repos
- develop branches created from the rel1713 branches (so normal development could continue during release certification)
- Once release is certified, a master branch will be created based upon all of the respective rel1713 branches.

Process Review

- Based on Vincent Driessen's post, A successful Git branching model.
- See Git-branching-model PDF.
- One additional constraint, rebase the following before merging:
  - Feature branches
  - Bugfix branches
- See Git-branching-model-revised PDF.

Tools

- Git-flow establishes some conventions to follow.
- The right tools can help ensure conventions are followed.
- gitflow-avh represents current evolution of original toolchain
  - Mac: `brew install git-flow-avh`
  - Ubuntu: `apt-get install git-flow`
    - Also see this page.
  - Windows: Install Git for Windows.
    - As of version 2.6.4, GitFlow (AVH edition) is included
- gitflow-avh is pretty flexible, but to help maintain our sanity, we should follow certain repo conventions, above-and-beyond what Vincent described.

Repo Conventions

```
$ git flow config
```

```
Branch name for production releases: master
Branch name for "next release" development: develop
Feature branch prefix: feature/
Bugfix branch prefix: bugfix/
Release branch prefix: release/
Hotfix branch prefix: hotfix/
✷ Support branch prefix: support/
Version tag prefix: rel-

✷ "support" lets you create/rebase a support/… branch from any commit
reference.
```

gitflow assumptions

- The shared repo remotes are labelled "origin"
  - Stash/Zimbra
  - GitHub/Zimbra
- You can still have as many other remotes as you like to facilitate team collaboration, etc.

Initializing Existing Repository

Make sure your local repo contains `master` and `develop` branches, then:

```
$ git flow init
```

```
Branch name for production releases: master
Branch name for "next release" development: develop
✷ Feature branch prefix: feature/
✷ Bugfix branch prefix: bugfix/
✷ Release branch prefix: release/
✷ Hotfix branch prefix: hotfix/
✷ Support branch prefix: support/
Version tag prefix: rel-

✷ Tool defaults
```

Feature branch example

- git flow feature start decouple-imap
  - creates "feature/decouple-imap" branch, based on "develop" branch
- git flow feature track decouple-imap
  - start tracking "feature/decouple-imap" that has been shared on "origin"
- git flow feature checkout decouple-imap
  - switch back to your local "feature/decouple-imap" branch
- git flow feature rebase decouple-imap
  - rebase your local "feature/decouple-imap" branch. Run with "-h" for more options!
- git flow feature publish decouple-imap
  - publish your changes back to origin
- git flow feature finish -r decouple-imap
  - rebase, then merge feature branch back to "develop" on origin

Going Forward

- **Release manager** uses standard tools to coordinate changes among all of our repos
  - Code freeze (initialize release integration branches)
  - Release certified (merge release integration branches into master and develop, tag master.
  - Other?
- Developers follow established conventions with working with branches
  - gitflow-avh can help

TODO

- Create command-line tools for release managers to use.
  - Greg has already started this effort. This helped with the 1713 release.
- Create a downloadable config file for gitflow-avh that automatic enables the rebasing option for features and bugfixes.
- Rename existing bug and feature branches to follow naming conventions.
- Get rid of any other "permanent" branches in the origins.