Git, JIRA, Crucible, etc.

MWF Project Workflow

Eric Bollens
ebollens AT ucla.edu
Mobile Web Framework Architect
UCLA Office of Information Technology

August 23, 2011
Overview

1. Introduction to the MWF Git Repository
2. Leveraging Git as a Service Provider
3. Development Process using JIRA & Crucible
Introduction
to the MWF Git Repository
Git

- Decentralized development
- Branch centered
- Local and remote
- Fast and efficient
- Flexible
- GitHub
MWF and Git

- MWF is available on Github
  - [https://github.com/ucla/mwf](https://github.com/ucla/mwf)

- Institutions can fork the MWF repository
  - Develop their own custom code
  - Contribute bug fixes back to the MWF
  - Merge in new version releases of the MWF

- Source-available license governs use
MWF and Git

- MWF
  - Institution
  - Institution
  - Institution
  - Developer
  - Developer
  - Developer
Git Workflows

- Git allows numerous workflows
- Several common models:
  - Centralized
  - Integration Manager
  - Dictator-Lieutenant
- More information:
Centralized

- Each developer commits locally
- Commits are pushed to a shared repository
- Conflicts must be resolved before push
Integration Manager

- Each developer maintains their own remote
- Request merges by integration manager
- Integration manager runs blessed repository
Dictator-Lieutenant

- Hierarchical with intermediate managers
- Each lieutenant represents some entity
- Allows for variation between entities
MWF Distributed Git Workflow

- Follows the Dictator-Lieutenant workflow
  - MWF core team manages a blessed repository
  - Entities using the MWF branch from it
- Each entity using the MWF
  - Has their own repository
  - Manages their own development processes
  - Implements their own custom code
  - Can contribute code back to MWF
MWF Distributed Git Workflow
MWF Distributed Git Workflow

- MWF Master
- MWF Core Team
- Institutional Master
- Institution Core Team
MWF Distributed Git Workflow
MWF Distributed Git Workflow

MWF Master

MWF Core Team

Institutional Master

Institution Core Team

Developer

Developer

Developer

Developer
MWF Development Git Workflow

- Branch-centric version control
- M.m.rr(.d) versioning scheme
- MWF core uses several types of branches:
  - master – Current production release
  - develop – Next production release
  - maint-[M.m] – Previous M.m release with bug fixes
  - feature/[name] – Feature development
MWF Development Git Workflow

1. Developer defines a specification
   – MWF participants review and propose changes

2. Developer creates a feature branch
   – Branched from version tag
   – Encapsulates an MWF feature or fix
MWF Development Git Workflow

3. When feature is complete, MWF core team
   – Performs code review and integration testing
   – Merges branch into develop
   – May also merge branch into maint-[M.m]

4. When develop is ready, MWF core team
   – Merges code to master
   – Creates a new version tag
MWF Development Git Workflow
MWF Development Git Workflow
MWF Development Git Workflow
MWF Development Git Workflow

- master
- develop
- feature/a
- feature/b
- tag
MWF Development Git Workflow

master

develop

feature/a

feature/b

tag
MWF Development Git Workflow

```
master  develop  feature/a  feature/b
   
  tag

```

The diagram illustrates the workflow where `master` is the main branch, `develop` is the development branch, and `feature/a` and `feature/b` are feature branches. The `tag` represents a tagged release or freeze point in the development process.
MWF Development Git Workflow
Leveraging Git
as a Service Provider
Service Provider Workflows

• Institutions can define their own workflow
  – UCLA uses the MWF development workflow
  – Different compositions warrant other workflows
• Institution can contribute back to MWF
  – Each institution has a lieutenant (point person)
  – Lieutenant coordinates contributions back
  – Git provides several contribution methods
Creating an Institutional Fork

1. Create a local repository
   - mkdir repo
   - cd repo
   - git init

2. Add institutional remote
   - git remote add origin [repository_uri]

3. Add MWF remote
   - git remote add upstream git@github.com:ucla/mwf.git
Creating an Institutional Fork

4. Pull master from upstream (MWF remote)
   – git pull upstream master

5. Push master to origin (institutional remote)
   – git push origin master

6. Pull tags from upstream (MWF remote)
   – git fetch upstream –tags
Creating an Institutional Fork

7. Branch the latest tag as master
   - git checkout -b [institution]/develop v[M.m.rr(.d)]
   - git checkout -b [institution]/master [institution]/develop

8. Push to origin (institutional remote)
   - git push origin [institution]/develop
   - git push origin [institution]/master
Creating an Institutional Fork

MWF Root Repository (Remote)

- upstream
- master
- tags

Institution Repository (Local)

- `git init`
- `git add remote origin ...`
- `git add remote upstream ...`
- `git pull upstream master`
- `git push origin master`

Institution Repository (Remote)

- `origin`
- `master`
- `[institution]/master`
- `[institution]/develop`

`git fetch upstream --tags`

- `git checkout -b [institution]/develop [v[M.m.rr(.d)]]`
- `git checkout -b [institution]/master [institution]/develop`
- `git push origin [institution]/develop`
- `git push origin [institution]/master`
Service Provider Development

• UCLA Mobile uses MWF workflow
  – Developers build modules as feature branches
  – UCLA Mobile team handles merges to develop
  – Development server runs develop branch
  – Tags represent production releases

• UCLA merges MWF tags into ucla/develop
  – Once integrated, merges into ucla/master
  – Tags master with similar M.m.rr(.d) scheme
Service Provider Development
Service Provider Development

develop  master

ucla/master  ucla/develop

MWF  UCLA
Service Provider Development

- develop
- master
- ucla/master
- ucla/develop
- tag

MWF  UCLA
Service Provider Development

develop  master

tag

tag

tag

MWF  UCLA

dev/develop  master  tag

tag
tag
tag

tag
tag
tag

ucla/master  ucla/develop  ucla/feature/a  ucla/feature/b
Service Provider Development

develop   master

ucla/master   ucla/develop   ucla/feature/a   ucla/feature/b

MWF   UCLA
Service Provider Development
Service Provider Development

MWF

UCLA
Service Provider Development

- develop
- master
- ucla/master
- ucla/develop
- ucla/feature/a
- ucla/feature/b

MWF

UCLA
Service Provider Development

MWF

UCLA
Service Provider Development

- develop
- master
- ucla/master
- ucla/develop
- ucla/feature/a
- ucla/feature/b

MWF

UCLA
Service Provider Development

MWF

UCLA

develop -> master

ucla/master -> ucla/develop

ucla/feature/a -> ucla/feature/b

tag

tag
Contributing Back to MWF

• Service provider can submit a patch to MWF
• Process parallels MWF core development
  – DEV: Propose the fix or feature *(optional)*
  – DEV: Develop an institutional feature branch
  – DEV: Submit merge request (JIRA)
  – MWF: Merge into an MWF branch
  – MWF: Code review the MWF branch (Crucible)
  – MWF: Merge into develop
Development Process
using JIRA and Crucible
JIRA

• Issue and task tracking for MWF project
• Used for
  – Bug reports
  – New feature development
  – Research and logistical tasks
  – Merge requests
• https://jira.ats.ucla.edu:8443
Crucible

• Code review system for MWF project
• Used for
  – All new features contributed into MWF
  – Large fixes and alterations of MWF
• [http://jira.ats.ucla.edu:8060](http://jira.ats.ucla.edu:8060)
Contributing Back to MWF

1. DEV: Create MWF “Merge Request” ticket
2. DEV: Add merge information:
   - If private repository, attach format-patch output
   - If public repository, include branch or revisions
3. MWF: creates branch and code review
4. DEV: Update based on code review
5. MWF: Merge to develop
Getting Access

- Request access from MWF core team
- All lieutenants should have access
- Open to any MWF user on request
Thank you for listening.