

EM&M Git Users Guide

Table of Contents

- **Overview**
 - Prerequisites
 - Notes:
- **Create new repo for the EM&M Git System**
 - Create temporary repo for sharing
 - Development lead initializes new EM&M Git repo
 - Finalize New Git repo
- **Copy repo from the EM&M Git System**
 - Copy an EM&M Git System repo to your workstation
 - Get Read only copy of EM&M Git System repo snapshot
- **Update local repo from the EM&M Git System**
- **Update the EM&M Git System from local repo**
 - First "Git push"
- **EM&M Git Repo System Work Flow**
 - The EM&M Branching and Merging Workflow
 - Gitolite Administration
 - Repository Branches
 - Main Branches
 - Supporting Branches
 - Feature Branches
 - Creating a feature branch
 - Merging a feature branch
 - **Development Deployment Branches**
 - Release Branches
 - Creating a release branch
 - Merging a release branch
 - Hotfix Branches
 - Creating the hotfix branch
 - Merging the hotfix branch
 - Release Tags
 - EM&M Modules in Git
- Quick References
 - Useful Command Syntax
 - Cheat Sheets
 - Useful Links

Overview

The EM&M Configuration Management (CM) team provides and supports the following for interacting with the [EM&M Git System](#). It is based on the [Git](#) tool. Refer to [About Git](#) for more information on the Git tool.

The EM&M Git System is managed with the [Gitolite](#) repository hosting tool. Some of the tools referenced below are provided via Gitolite configurations.

As development and unit testing occurs, most interaction with git will be through your local git repo on an isolated branch. The only times you will interact with the EM&M Git System will be when you:

1. Create new repo for the EM&M Git System.
2. Copy repo from the EM&M Git System.
3. Update local repo from the EM&M Git System.
4. Update the EM&M Git System from local repo.

Before you do any [git commits](#) on your local git repo, you will merge with the heads of EM&M managed branches to understand how your changes will work with the latest deployed code for your repo. This is the primary reason you will need to update your local git repo from the EM&M Git System. **Latest deployment information**, once established, is available from the EM&M Git System repo for:

- Development
- Release
- Hotfix
- Production

For more details on working with Git, review the [EM&M General Git Approach](#).

For a comparison of EM&M CVS usage and EM&M Git usage, review [Git Compared to CVS](#).

A combination of Git and ssh (via gitolite) commands are available for interacting with the EM&M Git System. Reference [Sharing and Updating Projects](#) from the [Git Reference](#) site for an overview of the available git commands.

Note: This reference covers the Git "Porcelain" commands that are build on the Git "Plumbing" commands. Reference [Git Internals - Plumbing and Porcelain](#) for details.

The following sections provide examples of anticipated sessions, with syntax, you are likely to encounter while working with the EM&M Git System.

Prerequisites

In order to use the commands in this document:

1. Complete the [Git Installation Procedure](#).
2. Your UserID (ssh key) must be in the proper gitolite group or ROLE for repo access.
3. Refer to the [EMM GitWeb](#) for:
 - a. Repo name reference.
 - b. Heads Section (under summary) for branch names.
 - c. tmp - This directory indicates repos created here are temporary in nature. Once they have been merged with devlead owned repos the developer has the EMM CM team remove them. If repo not active for more than six months, it will be removed.
 - d. CREATOR - This is substituted with the analyst's logon ID as
identified with the analyst's public ssh key name.
 - e. LOB - Line of Business, one of CFX, VIDEO, VOICE, or DATA
 - f. Module - Name at discretion of Dev Lead. Does not contain / or _.
 - g. Major - Number for baseline functionality.
 - h. Minor - Number for functionality changes still functional with associated Major number.

Notes:

1. EM&M Git System Modules are identified based on the [EM&M Module Standards](#).
2. Initially, there is no default master branch on the new EM&M Git System temporary repos. You can create one for your local repo.
 - a. In order to populate the staging area of your local repo on your workstation for repos with no **master** branch, either:
 - i. Identify the branch to checkout on the [Git clone](#) or
 - ii. [Git checkout](#) to a branch on your local repo copy on your workstation after you clone it from the EM&M Git System.
3. You will not be able to [Git push](#) to the remote (EM&M Git System repo) **master**, **hotfix**, **release**, or **develop** branches.
4. You must be a developer lead to [Git push](#) to the remote (EM&M Git System repo) **develop** branch.
5. You will not be able to create a tag that starts with **v[0-9]**. This is reserved for EM&M Git System Production Releases.
6. For any git command, you can enter git [command] -h for command documentation.
7. EM&M Git System repo branch and tag access rules are configured and maintained in the gitolite-admin repo, conf/gitolite.conf file.
8. The [Git Reference](#) site is a great quick reference for git "Porcelain" commands. Reference [Git Internals - Plumbing and Porcelain](#) for details.

Create new repo for the EM&M Git System

Create temporary repo for sharing

Sharing of Git repo branches in established EM&M Git System repos is probably the easiest way for developers to share code using the [EM&M Git System](#).

If there is a need to establish a temporary repo to share with other developers, the following has been provided. Just as temporary repo branches should be removed when no longer used, so to these temporary Git repos should be removed when no longer needed.

If either, temporary repo branch or temporary repo, has not been used for more than six months, they will be removed from the EM&M Git System.

These repos are not backed up to the EMM System failover server. Code will need to be merged with DevLead managed repos for deployment outside of development.

Syntax	Comments
<pre>git clone emm-git:tmp/CREATOR/[LOB]-[Module]-[SubModule] cd [LOB]-[Module]-[subModule] git checkout -b [branch name] Add your files, git add and commit ... git push --set-upstream origin [branch name]</pre>	<p>CREATOR is substituted with your Comcast userid (name of your public ssh key less .pub).</p> <p>In order to share files, establish a repo on the EM&M system and your workstation, create a branch, add files to that branch with git add and commit, set the branch in the EM&M git repo pushing your files to it.</p>

Development lead initializes new EM&M Git repo

A development lead will exercise this procedure for new code sets. These repos will need to be Finalized, see next section, before they can be used to contribute to the [EM&M Deployment WorkFlow](#).

These repos are not backed up to the EMM System failover server. The EM&M Git System Repo must be Finalized by CM staff. All users of pre-Finalized repos need to re-clone their local copy of the repo from the finalized location in the EMM Git System.

Syntax	Comments
<pre>\$ git clone emm-git:CREATOR/[LOB]-[Project] Cloning into 'repo name'... WARNING: This system is solely for the use of authorized Comcast employees and contractors. Initialized empty Git repository in /app/git-repos/[userid]/[repo name].git/ warning: You appear to have cloned an empty repository. Checking connectivity... done. \$</pre>	This will create a repo in the EM&M Git System and create a copy of it on the executors workstation.
<pre>\$ cd [repo name]</pre>	Get local to the repository directory on your workstation.
<pre>\$ git branch -v develop master * test-branch</pre>	List existing branches in this repo. The "*" indicates the branch currently in your Staging area. Reference The Three States for details.
<pre>\$ git checkout develop Switched to branch 'develop'</pre>	Switch to an existing develop branch.
<pre>\$ git checkout -b develop</pre>	Create and switch to a new develop branch.
<pre>\$ \$ git add . \$ git commit -m "[Comment of your choosing.]" [develop (root-commit) 15e8e6f] [Comment of your choosing.] \ N file changed, N insertion(+) create mode 100644 filename create mode 100644 filename2 ... \ \$</pre>	Add/move files to this directory. Use "git status" as needed

Finalize New Git repo

For Git repos in the [EM&M Git System](#) there are [Module Standards](#), requirements for [EM&M GitWeb](#), and [Gitolite](#) configurations that need to be made. A [CHANGELOG](#) is also required. Once these modifications are completed for a given Git repo in the EM&M Git System, the repo is ready for tracking all relating coding and deployments for EM&M. It is "Finalized" at this point.

All module repo names starting with tmp/[userid] or [userid]/ on the [EMM GitWeb](#) are newer repos that have not been finalized. Before a Git repo gets deployed to a EM&M deployed to environment, it needs to be Finalized.

The following table contains syntax that introduces gitolite configured functionality available to EMM Git System repos.

Syntax	Comments
--------	----------

<pre> \$ ssh git@emm-git help WARNING: This system is solely for the use of authorized Comcast employees and contractors. hello awalla5075k, this is git@emmutl-po-7p running gitlite3 (v3.6.1) on git 1.9.0 \ List of remote commands available: \ desc help info perms writable \ \$ ssh git@emm-git desc WARNING: This system is solely for the use of authorized Comcast employees and contractors. \ Usage: ssh git@host desc <repo> ssh git@host desc <repo> <description string> \ \$ </pre>	<p>Users of this EMM Git repository have direct access to the repository server via this defined ssh session</p> <p>In addition to regular git remote repository commands. The specific "remote command" help detail as indicated in the second syntax example provides specific syntax for each of the remote commands listed.</p> <p>\</p> <p>As a repo owner/creator, you can manage users of a given repo via the perms option for the following roles:</p> <ul style="list-style-type: none"> • OWNERS • WRITERS • READERS • MSLEADS - Managed Service Leads • MSDEVS - Managed Service Developers
---	---

Reference [Gitolite](#), the [User's View](#). This reference provides an introduction of functionality available to the users of the EMM Git Repository System.

To request finalization of a git repo, Sent and email to the EMM CM Team (Robert Sell, Andrew Wallace, Bruce Woodcock) that identifies the EM&M Git repo you wish to finalize.

All users of pre-Finalized repos will need to re-clone their local copy of the repo from the finalized location in the EMM Git System.

Copy repo from the EM&M Git System

You must have an established ssh key with the [EM&M Git System](#) for this to work. refer to [Build and Configure SSH Key Usage](#) for details.

Copy an EM&M Git System repo to your workstation

Syntax	Comments
<code>git clone emm-git:[repo name]</code>	Copy an EM&M Git System repo to your workstation with workfiles set to the head of the default master branch.
<code>git clone -b [branch name] emm-git:[repo name]</code>	Copy an EM&M Git System repo to your workstation with workfiles set to the head of a given branch.

Get Read only copy of EM&M Git System repo snapshot

This will not create a Git repo. It does provide the state of all files for a given [Git commit](#). This is referred to as a [Git snapshot](#). Using a snapshot can be used to get a new developer working on a code set quickly. The only requirement is a unix or linux shell.

Instructions	Comments
Go to EMM GitWeb , select desired module, select snapshot of desired branch/tag.	This will provide a gzip file that you can install on your workstation.

Update local repo from the EM&M Git System

Review [Sharing and Updating projects](#).

Syntax	Comments
<code>\$ git fetch</code>	Reference http://gitref.org/remotes/#fetch .
<code>\$ git pull</code>	Reference http://gitref.org/remotes/#fetch . Caution, this is a fetch followed by a merge.

Update the EM&M Git System from local repo

Syntax	Comments
<code>\$ git push --set-upstream origin develop</code>	Push the contents of the Git develop branch in the local Git repo on your workstation to the EM&M Git System. The argument "--set-upstream" required to initiate the new develop branch on the EM&M Git System. <i>You must be a devlead to execute this command.</i>
<code>\$ git push</code>	Requires local repo to have been setup with a "git clone" option that has write authority on the EM&M Git server.

First "Git push"

Before executing a "git push" for a given EM&M Git System repo remote, the following questions should be considered:

- Did you Review [The Three States](#)?
- Are you familiar with the [Git Reference](#) site?
- Are you familiar with the "**Git Approach**"? see **Notes:** (3rd note) in [Overview](#) section of this document.
- What [git branch](#) are you attempting to update for the repo in question?
- What [git remote](#) are you using for your "git push"?
- Did you configure your [CHANGELOG](#) for the Development (**develop** branch) completed? (Branch entry in Build ID:)
- Did you develop on a [feature branch](#) and merge your changes to your development team's [supporting branches](#) on the EM&M Git System?
- Did you test and verify your [git merges](#) with the EM&M git repo production (master) and develop branches before using "git push" to your development team's EM&M Git System [supporting branches](#)?
- Did you remove your [feature branch](#) once it was successfully merged to a development team [supporting branch](#)?
- Are you allowed to "git push" to the remote branch on the EM&M Git System primary server where you are trying to find your changes?
 - Note that the **develop** and **master** [main branches](#) on the EM&M Git System repos have restrictions.

The utilities available to you when considering the above questions:

(Be local to your repo on your workstation for the following "git" commands. This is not necessary for the "ssh" commands.)

\$ git status - Reports on current state of your local repo.

\$ git branch - Lists the current branches in your git repo.

\$ git remote - Lists the configured remotes for your git repo. The origin remote is automatically setup for you when you create or acquire a git repo from the EM&M Git System.

\$ git remote -v show origin - For your git repo origin remote, shows the git fetch and push URLs, tracked remote branches for your branches and branch push statuses.

\$ ssh git@emm-git info - Shows your access to EM&M Git System repos. Reference `ssh git@emm-git help` for details.

\$ ssh git@emm-git perms -l [repo name] - Shows Roles the repo owner has granted you on the repo. One of OWNERS, WRITERS, READERS, MSLEADS, MSDEVS.

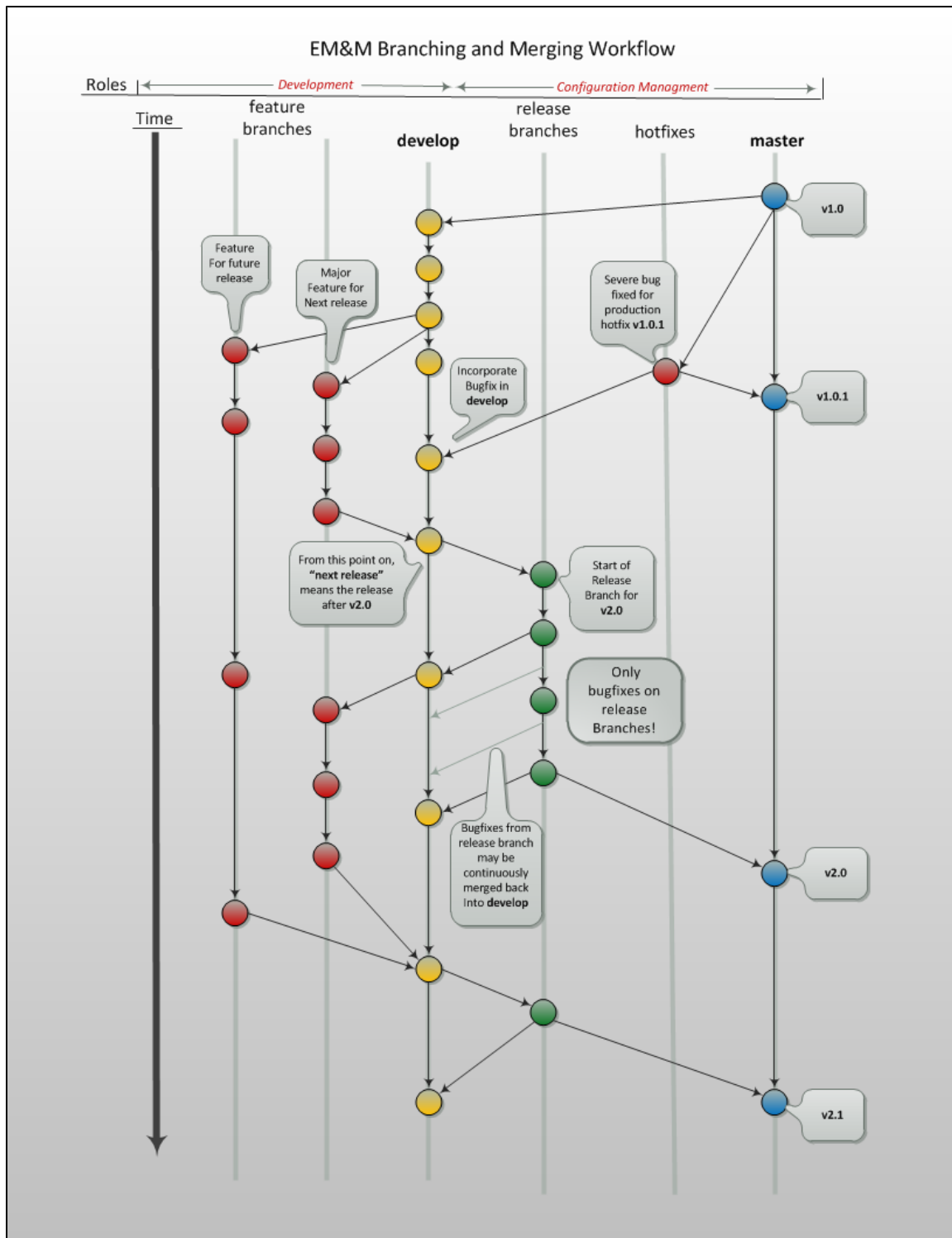
Did you review the [Notes:](#) section in the [Overview](#) section of this document?

EM&M Git Repo System Work Flow

The primary reference for EM&M Git documentation is <http://git-scm.com/documentation>. It is recommended you cover "Git Basics" and "Git Branching" at minimum. It is highly recommended you review the [Git Reference](#) before moving further in this document. This review provides a basis of Git understanding that is assumed for the remainder of this User Guide.

In order to understand how branching and merging is utilized for the EM&M Change and Release Workflow review [A Successful Git Branching Model by Vincent Driessen](#) in it's entirety. EM&M specifics are covered in following sections of this document.

The EM&M Branching and Merging Workflow



Gitolite Administration

The application Gitolite is being used to administer this server by the Configuration Management team. See [What Users Should Know about Gitolite](#) for details.

Repository Branches

The first task a user of the EM&M repository system does is to create a branch of a module repo branch they are targeting for change. Before any change is introduced back into the EM&M Git System, the user must merge their change into the appropriate Main Branch of the targeted module repo on their local git repo. This is indicated in the [EM&M Branching and Merging Workflow](#). Module repo leads are assigned to aid users with

these branching and merging tasks.

Main Branches

Main Branches are permanently maintained. Each "git commit" and "git push" to a main branch is for code sets that build automatically. Development maintains the **develop** branch that is fully automated for builds and deployments to development environments. The Configuration Management and the Event Processing Assurance (EPA) teams maintain the **master** branch for production ready and production installed deployments. The master branch is used to initialize **hotfix** branches. These **hotfix** branches are used to promote production problem related changes to **master** branch and introduce these changes to the **develop** branch.

"Merge Points" in the EM&M Git System are kept to a minimum to keep collaboration consistent and simple. The two main merge points provided for EM&M development by the EM&M Git System are "Latest development" and "Latest Production". The "Latest Development" is buildable and deployable to the development environment. The "Latest Production" is the default branch for production ready code.

There are two Main Branches in any given EM&M git repo on the repository system, **develop** and **master**. These branches are intended for deliverables that build and deploy in an automated fashion. In the case of the **develop** branch, each "git commit" and "git push" should automatically build and deploy to the development environment(s). For the **master** branch, each "git commit" and "git push" should automatically build and provide for "on demand" deployment to production environments. In addition, the **master** branch is only updated with "Production Ready" deliverables as determined from development and testing activities. Supporting branches are used to manage branching and merging tasks performed before and between these Main Branches.

Notes:

1. In "Git Speak", **origin** is associated with the EM&M Git System repo server.
2. Each EM&M Git System repo supports a single production installation. If multiple production installations are required, multiple EM&M Git System repos are maintained with repo name and dependency tracking.

The **develop** branch is the primary integration branch where automatic builds occur. The source code in the **develop** branch head is a stable point ready to be released. The **release** branches are established from the **develop** branch head. The **release** branch is used to prepare for the next production release. The **release** branch identifies code that development has deemed "Production Ready". The **master** branch is kept in sync with the latest production deployed release.

Supporting Branches

There are three types of supporting branches, **feature**, **release**, and **hotfix**.

Development teams also maintain "supporting branches" for coordinating development efforts. For example, development teams may coordinate several feature branches into development team managed release branches. These branches can be established in EM&M Git System repos at the development team's discretion. Check with your Development Lead for supporting branches used by your team.

Feature Branches

Should branch off from: develop

Must merge back into: develop

Branch naming convention: Change Record or User Story number with developer deployment tags, (see below), or temporary name. Never master, develop, release*, or hotfix*

Feature branches (or sometimes called topic branches) are used to develop new features for the upcoming or a distant future release. When starting development of a feature, the target release in which this feature will be incorporated may well be unknown at that point. The essence of a feature branch is that it exists as long as the feature is in development, but will eventually be merged back into develop (to definitely add the new feature to the upcoming release) or discarded (in case of a disappointing experiment).

Feature branches typically exist in developer repos only, not in origin (EM&M Git System server). When merging a feature branch into the development branch, it must be coordinated with the assigned development lead.

Creating a feature branch

When starting work on a new feature, you can branch off from the develop branch as follows:

```
$ git checkout -b USfeature1 develop
Switched to a new branch 'USfeature1'
$
```

Merging a feature branch

Finished features must be merged into the develop branch in order to add them to the upcoming release:

```

$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff USFeature1
Merge made by the 'recursive' strategy.
 License      | 3 +++
 readme.txt   | 2 ++
 test.rb      | 2 ++
 3 files changed, 7 insertions(+)
$ git branch -d USFeature1
Deleted branch USFeature1 (was 6786061).
$ git push origin develop
(summary of change)
$

```

The --no-ff flag causes the merge to always create a new commit object, even if the merge could be performed with a fast-forward. This avoids losing information about the historical existence of a feature branch. It groups together all commits that together added the feature.

Development Deployment Branches

In this organization, development work requirements are done with Rally User Stories. These User Stories are associated with JIRA tickets that are maintained in the JIRA CEMPCM project workflow. The title of JIRA tickets is maintained in the summary field of the JIRA ticket with the Rally User Story number and the module release identifier. The Development Deployment Branch name is the module release identifier portion of the JIRA summary field. The Module name is the repo name or one of many repos related to a given Module. It format is:

- [ModuleName]_N_N_N_N

The ModuleName format is as follows (Refer to the [EMM Module Standards](#) for details):

- LOB - Line of Business (CFX, DATA, VIDEO, VOICE).
- Module - Examples include AUPM, EST, UID, CDV. (Project Level)
- SubModule - Examples include MED, CON, COLL, CABS. (Application Group level)
- SubModule - Examples include Pub. (Application level)
 - **NOTE:** Product Line (PL), the second element of older module standard for CVS is being dropped as we move to Git.

Format of ModuleName = LOB-Module-SubModule-SubModule

NOTES:

1. In the ModuleName the "-" is changing to the "_".
2. The Product Line (Business Residential, one or the other or both) is being dropped.
3. Modules are defined to a level where each module is a baseline deployment.

The **source of record** for requirements is Rally. The **source of record** for deployments is the JIRA CEMPCM tickets. When developers work on and deliver EM&M deployments for release outside of development to the QA, Integration, Staging, and Production environments, they do it with developer deployment branches. These are **feature branches**. The names of these deployment branches are the same as the module release identifier used in the JIRA ticket summary field.

When the developer deployment branch is buildable and deployable to the development environment, it is merged with the EM&M git system **Develop** branch. **This is a requirement for future build and deployment automation.**

NOTE: The git commit immediately following the git merge **MUST** include the name of the Development Deployment Branch found in the JIRA summary field.

The format of the developer deployment branch name is [ModuleName]_N_N_N_N where each "N" is a numeric increment for Major, Minor, Dev, Test. EM&M Intake system record numbers and Deployment branches are used to manage EM&M deployments from development, to QA, Test, and Production environments. The [EM&M Workflow](#) describes how this is managed.

- **NOTE:** When all developer deployments are buildable and deployable before they are merged to the **develop** branch, the "Dev" and "Test" increments should be re-considered.

The EM&M Module Release Tag identifies a specific Deployment set that is developed, tested, and released to EM&M QA, Test, and Production environments. Each number in the Module Release Tag has a specific meaning to identify specific set of elements in support of the EM&M Workflow for design, development, test, deploy, and production deployment.

- **Major** - A numeric integer used when tagging a module. A major module identifier typically indicates no backward compatibility with previous decremented major values. MAJOR version numbers are designed to be incremented for every full release of the module.
- **Minor** - A numeric integer that indicates a partial release to most recent MAJOR_MINOR release set. For example, release "2_22" would represent a delta that is added to release "2_21". Dependency to the previous MINOR number with a common MAJOR version number is

assumed. Any other dependencies are documented in the install or design documents by the developer.

- **DEV** - Numeric increment for each version of the MAJOR_MINOR release delivered by EM&M Development, initially set to "0" on first delivery of a Major_Minor identified module. If requirement change or development needs to deliver another version, this number is incremented, and the Test number is set to "0". Separate instructions from the install doc can be included with the build request for the QA environment. This would account for deltas that have changed since the QA install has already been done in the previous version.
- **Test** - Numeric increment for each version of the MAJOR_MINOR_DEV release delivered by EM&M Development, initially set to "0". If EM&M QA/Test drafts a defect for a MAJOR_MINOR_DEV_Test delivered and installed, the "Test" value is incremented by development for the next delivery addressing the defect. Separate instructions from the install doc can be included with the build request for the QA environment. This would account for deltas that have changed since the QA install has already been done in the previous version.

At this point the automated build is performed to validate all automated Build and Unit Test procedures are functioning successfully.

Release Branches

Should branch off from: develop

Must merge back into: develop and master

Branch naming convention: release-[Major].[Minor]

Note: - Major and Minor as defined below in the "Development Deployment Tags" section.

Release branches support preparation of a new production release. They allow for last-minute dotting of i's and crossing t's. Furthermore, they allow for minor bug fixes and preparing meta-data for a release (version number, build dates, etc.). By doing all of this work on a release branch, the develop branch is cleared to receive features for the next big release.

The key moment to branch off a new release branch from develop is when develop (almost) reflects the desired state of the new release. At least all features that are targeted for the release-to-be-built must be merged in to develop at this point in time. All features targeted at future releases may not---they must wait until after the release branch is branched off.

It is exactly at the start of a release branch that the upcoming release gets assigned a version number---not any earlier. Up until that moment, the develop branch reflected changes for the "next release", but it is unclear whether that "next release" will eventually become a new minor or major release until the release branch is started. That decision is made on the start of the release branch and is carried out by the project's rules on version number bumping.

When merging a release branch into the development branch, it must be coordinated with the assigned development lead. When merging a release branch into the master branch, it is performed by the Configuration Management team, coordinated with the affected development leads, and coordinated with the EM&M release team and Comcast National Change Management (NCM) via an NCM CM and a EM&M "Go No Go" meeting.

Creating a release branch

Release branches are created from the develop branch. For example, say version 1.1.5 is the current production release and we have a big release coming up. The state of develop is ready for the "next release" and we have decided that this will become version 1.2 (rather than 1.1.6 or 2.0). So we branch off and give the release branch a name reflecting the new version number:

```
$ git checkout -b release-1.2 develop
Switched to a new branch "release-1.2"
$ ./bump-version.sh 1.2
Files modified successfully, version bumped
to 1.2.
$ git commit -a -m "Bumped version number to
1.2"
[release-1.2 74d9424] Bumped version number
to 1.2
1 files changed, 1 insertions(+), 1
deletions(-)
$
```

After creating a new branch and switching to it, we bump the version number. Here, bump-version.sh is a fictional shell script that changes some files in the working copy to reflect the new version. (This can of course be a manual change---the point being that some files change.) Then, the bumped version number is committed.

This new branch may exist there for a while, until the release is rolled out in the Production Environment. During that time, bug fixes may be applied in this branch (as well as the develop branch). **Adding large new features here is strictly prohibited. They must be merged into the develop branch, and therefore, wait for the next scheduled release.**

Merging a release branch

When the state of the release branch is ready to become a real release, some actions need to be carried out. First, the release branch is merged into master, since every commit on master is a new release by definition. Next, that commit on master must be tagged for easy future reference to this historical version. Finally, the changes made on the release branch need to be merged back into the develop branch so that future releases will also contain any applied bug fixes.

The first two steps:

```
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff release-1.2
Merge made by the 'recursive' strategy.
 License      | 3 +++
 readme.txt   | 2 ++
 test.rb      | 2 ++
 3 files changed, 7 insertions(+)
$
```

The release is now done, and tagged for future reference.

Edit: You might as well want to use the -s or -u <key> flags to sign your tag cryptographically.

To keep the changes made in the release branch, we need to merge those back into the develop branch:

```
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff release-1.2
Merge made by recursive.
(Summary of changes)
$
```

This step may well lead to a merge conflict. If so, fix it and commit.

Now we are really done and the release branch may be removed, since we don't need it anymore:

```
$ git branch -d release-1.2
Deleted branch release-1.2 (was 41a3f00).
$
```

Hotfix Branches

Should branch off from: master

Must merge back into: develop and master

Branch naming convention: hotfix-[Major].[Minor].[hfx#]

Notes:

1. If there is an active, in-progress release branch, merging should be considered depending on current state of major branches.
2. hfx# - HotFix number. Increments for each hotfix of a given Major.Minor release.

Hotfix branches are very much like release branches in that they are also meant to prepare for a new production release, albeit unplanned. They arise from the necessity to act immediately upon an undesired state of a live production version. When a critical bug in a production version must be resolved immediately, a hotfix branch may be branched off from the corresponding tag on the master branch that marks the production version.

The essence is that work of team members (on the develop branch) can continue, while another person is preparing a quick production fix.

When merging a hotfix branch into the development branch, it must be coordinated with the assigned development lead. When merging a hotfix

branch into the master branch, it is performed by the Configuration Management team, coordinated with the affected development leads, and coordinated with the EM&M release team and Comcast National Change Management (NCM) via an NCM CM and a EM&M "Go No Go" meeting. If there is a release branch in progress at the time the hotfix is merged into the master branch, the hotfix may also need to be merged into the release branch depending on the current release in progress circumstances.

Creating the hotfix branch

Hotfix branches are created from the master branch. For example, say version 1.2 is the current production release running live and causing troubles due to a severe bug. The changes on the develop branch are currently unstable. We may then branch off a hotfix branch and start fixing the problem:

```
$ git checkout -b hotfix-1.2.1 master
Switched to a new branch "hotfix-1.2.1"
$ ./bump-version.sh 1.2.1
Files modified successfully, version bumped
to 1.2.1.
$ git commit -a -m "Bumped version number to
1.2.1"
[hotfix-1.2.1 41e61bb] Bumped version number
to 1.2.1
1 files changed, 1 insertions(+), 1
deletions(-)
$
```

Don't forget to bump the version number after branching off!

Then, fix the bug and commit the fix in one or more separate commits.

```
$ git commit -m "Fixed severe production
problem(hotfix-1.2.1)"
[hotfix-1.2.1 cd3ebfe] Fixed severe
production problem(hotfix-1.2.1)
1 file changed, 1 insertion(+)
$
```

Merging the hotfix branch

When finished, the bugfix needs to be merged back into the master branch, but also needs to be merged back into the develop branch. This ensures the bugfix is included in the next release. This is completely similar to how release branches are finished.

First, update master and tag the release.

```
$ git checkout master
Switched to branch 'master'
$ git merge --no-ff hotfix-1.2.1
Merge made by the 'recursive' strategy.
License | 1 +
1 file changed, 1 insertion(+)
$ git tag -a 1.2.1 -m "new tag"
$
```

Edit: You might as well want to use the -s or -u <key> flags to sign your tag cryptographically.

Next, include the bugfix in develop, too:

```
$ git checkout develop
Switched to branch 'develop'
$ git merge --no-ff hotfix-1.2.1
Merge made by the 'recursive' strategy.
 License | 1 +
 1 file changed, 1 insertion(+)
$
```

The one exception to the rule here is that, when a release branch currently exists, the hotfix changes need to be merged into that release branch, instead of the develop branch. Back-merging the bugfix into the release branch will eventually result in the bugfix being merged into develop too, when the release branch is finished. (If work in the develop branch requires this bugfix and cannot wait for the release branch to be finished, you can merge the bugfix into the develop branch immediately.)

Finally, remove the temporary branch:

```
$ git branch -d hotfix-1.2.1
Deleted branch hotfix-1.2.1 (was cd3ebfe).
$
```

Please be sure you have reviewed the Standards section above in the Overview section of this document before moving on from this point.

Release Tags

Release and HotFix branches are tagged at the time they are deployed for future historical reference purposes. These tags are used to identify the current Repo Release in the Production Environment and the HotFixes association to each Repo Release.

The components of the Release Tags are as follows:

Since the **ModuleName** and the repo name are synonymous, the release tag starts with a "v" indicating version of the repo.

Major - A numeric integer used when tagging a release. A major release identifier typically indicates no backward compatibility with previous decremented major values. MAJOR version numbers are designed to be incremented for identified full release.

Minor - A numeric integer used when tagging a minor release. A minor release identifier typically indicates backward compatibility with previous decremented minor values. MINOR version numbers are designed to be incremented for identified minor releases.

HotFix# - Number incremented for each HotFix applied to a given Major.Minor release.

Note: - A release tag composed of Major.Minor with no HotFix number or "0" indicates no HotFix has been applied.

EM&M Modules in Git

In this organization, a Git repository **Module Name** consists of the following elements:

- **LOB** - Line of Business, one of CFX, DATA, VOICE, and VIDEO
- **Project** - Examples include AccountManagement, ADOPTOUT, AMDOCSOUT, ASTRO, AuditService, AUPM, AutoGen, BEACON, BlockingService, BPM, CANOE, CDV, CEMP, CIMCO, CLK, CloudUI, CMS, COLUMBUS, ContractAutomation, CPORTAL, CSG, CustMove, CycleChange, DDS, EEG, EEP, EEPBill, EMARS, EM&M, EST, FRAUD, HSD, HSI, IMS, ITV, JANUS, LCR, LDC, LEGAL, MANILA, NASR, NGT, ODS, OSM, PREPAID, RAZOR, SAP, SAVILLE, SDV, TVE, UES, UID, VAPI, VCME, VGD, VODGift, WaterMark, WHOLESALE, WIFI, WLS, XTM

Quick References


Useful Command Syntax

Syntax	Comments
\$ git diff master origin/master	After "git fetch origin", shows difference between master branch and branch on emm-git repository (origin).

<code>\$ git log --pretty=oneline --graph --decorate</code>	Report representation of current repo. (May need to send to tmp file to see full report - >> tmp.txt)
<code>\$ gitk</code>	Visual representation of current repo.

Cheat Sheets

Following are "Cheat Sheets" of tools you will use often in this environment:

vim	git simple	git	git
			

Git Cheat Sheet by Jan Krueger.

Git Cheat Sheet by Zack Rusin.

Useful Links

Commit Often, Perfect Later, Publish Once: Git Best Practices
Undoing Things
Git Software
Git Operations
Git reference
The Git Parable
Git from the bottom up
Git for Computer Scientists
Understanding Git Conceptually
git fetch and merge, don't pull
Git Tutorial
Eclipse Git Tutorial
Git - The simple guide
Why Managed Git Deployment
Cloudways - Using Git for Deployment