EM&M Git Users Guide

Table of Contents

Overview

The intended audience for this User Guide is all roles active in the Event Management & Mediation (EM&M), (formerly Converged Event Management Platform (CEMP)), organization. The associated admin guide for this user guide is the Git Repo System Admin Guide.

The EM&M Git Primary Repository server alias is emm-git1.sys.comcast.net.

All the sections of this document leading up to the **Documentation** section are intended for setting up and getting oriented to the EM&M development environment based on the Git tool.

If you have not worked with Git before, it is 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.

The purpose of this document is to:

1. Identify how to set up and maintain a Git environment on EM&M development and analyst Windows 7 workstations.
2. Provide a single document for all things related to EM&M Git usage. This includes related Standards and Deployment rules.
3. Provide introductions and links to appropriate documents for more detailed reference.

Notes:

1. When using links on this page, **Right-click** and select **Open link in new tab** to maintain visibility to this page.
2. The Git tool is not file-oriented, each check-in can be a set of many files.
3. The Git approach:
    a. Branch from a known managed branch,
    b. Adjust and modify your branch for needed changes,
    c. Merge your branch into all known managed branches,
    d. Test all branches you merged into for the new changes.
4. Where did the term Git originate from? Reference the **History** section of Git (software) in Wikipedia.

Standards

The following standards have been identified and are maintained to support smooth, automated release deployments to EM&M production environments. Every effort is made to identify the minimum set of standards required to keep the complexity of deployment process and procedures to a minimum.

The ability to merge changes with existing deployments and development sets before an environment is deployed to is essential for build and deployment automation and stable environments. The environments this system is used to support must be fully represented in the deployment sets new changes are merged with. This means that at the time an environment is deployed to from this system, steps must be taken to ensure the deployments, by environment, match the deployment sets, with and understanding of all remaining deltas, maintained in this Distributed Version Control System (DVCS).

If the deployments are **Full Module Baselines** [Note 1], this is easy to maintain at the time of environment deployments. If changes are allowed in the deployed-to environments without adhering to these standards, steps need to be taken at environment deployment time to ensure the DVCS maintained deployments match the environments they are intended for. If this is not done, development merges will be incomplete for future deployments causing future deployment failures. While **Full Module Baseline** [Note 1] can be automated for builds and deployments, these steps to ensure all environments changes are accounted for are not without human intervention. Unanticipated environment changes can cause deployment failures even if they are identified before a deployment is allowed to complete. This is the main reason automation is difficult, however, not impossible for these steps.

It is best to know about and plan for all environment changes prior to environment deployments. This can be accomplished by only allowing changes to supported environments via this DVCS.

Identify Dependencies

- EM&M Release tag that the current Developer deployment replaces
- Release tag of the Developer deployment tag this module was last released to the Production environment with
- Data Streams
- Database bases by environment, tables by database, rows by tables, fields by rows and tables
- Dependent applications are listed by deployment or release identifier starting with OS deployment/release
- List all environment variances for Development, QA, Test, and Production environments. Identify why they are currently needed and what the current mitigation plans are for each variance.

Deployment Rules

All deployments are the result of branching and merging activities as described in this documentation. It is imperative that merges be reviewed and performed by developers or lead developers familiar with all deltas involved with the merge. It is the responsibility of each developer performing a merge to seek out help for all deltas they may not be familiar with.

For consistent reference of source and deployment elements, it is recommended you have the following higher level directories in your repo file structure:

- app - Directory for all elements related to the application including build structure and required libraries, etc.
- db - All database related elements.
- doc - All documentation for deployment related activities, build instructions, environment validation documents, etc.

Deployments are **full module baselines** [Note 1] to support automated builds and deployments. The target on the system is comprised of the module name (repo name) and the release identifier (Tag). This allows for testing and comparison activities on the target system.

All dependencies for a given repo release deployment are identified and verified in the target environment at the time of the deployment. This is done with a CHANGELOG file that is maintained for each module repo. For example, the Gitolite project maintains this CHANGELOG.
CHANGELOG is currently being defined and coordinated.

If a portion of the full module baseline is the target of an environment install:

- All elements of the target deployment MUST be identified and scripted for each target environment deployment.
- All **variance** [Note 2] between the repo tagged elements for the release and deployed elements in the environment are identified and cataloged.
- All identified variances are either checked into the repo or adjusted in the environment. This is how environment-deployed elements are merged with the repo release tagged elements.
  Note: - If the deployment is a Full Module Baseline, the bullet tasks above are not required.

Quick-fix patch deployments are based on hotfix branches. The hotfix branching process is described below in this document.

No deployments are made to any of the EM&M environments (QA, Test, UAT, Production, etc) unless they are managed by the above EM&M deployment rules.

Notes:

1. - **Full Module Baselines** are all release-tagged elements that are a part of, or contribute to, the deployment set. The deployment set is a full representation of the Module being deployed. This includes all elements that changes can be made to for the module in question.

Conditionally, source code files in the repo are tagged with the release tag they contribute to. Resulting executables from tagged source file set builds are deployed to the environment. This deployment is done under a structure that identifies the repo release and are not in the repo or tagged since they are objects from the build that uses the source code files as input. Also included are tagged database file revisions maintained in the repo that implement or describe the structure of the Database(s) implemented and maintained with the repo.

2. - For variance identification, if a deployed element in the target environment is found to be different that its release-tagged counterpart in the repo, or is not in the repo, it is deemed a variance. If a release-tagged element is not found in the deployed environment or is different that it's deployed counterpart in the environment, it is deemed a variance.

Prerequisites

- Comcast workstation/laptop with Windows XP or 7
- Comcast NTLogon with admin access on workstation
- Each analyst needs to install git. See the Installing Git section of this document.

Installing Git

Click on the **DOWNLOADS** link at Git for Windows, download the latest "Full installer for official Git for Windows" release, (latest beta), and run it on your workstation system to install. You will see several, use the latest one.

- During the install you will encounter the following windows along with other standard install windows. Use defaults settings/options along with the selections indicated in the following windows: (Click on the windows to enlarge)

First-Time Git Setup

After you have installed Git successfully, there are a few tasks you need to perform in order to use git in the EM&M organization. You need to:

- setup GitHome on Your Workstation,
- identify yourself in Git,
- setup your default editor,
- setup the diff tool you will use.

Identify the $HOME variable Your Workstation

1. Click on Start (Lower Right Windows Bubble)
2. Right Click on **Computer** and select **Properties**
3. Select **Advanced system settings**
4. Click on the **Environment Variables...** button on the "Advanced" tab of the System Properties window
5. Identify and take note of the value of the **HOME** variable.
    a. Should be something like C:\Users[your NTLogon].
    b. This variable value will be utilized by the SSH key to be setup.

Identify Yourself in Git, Default Editor, and Diff Tool

1. Enter your first and last name:
   $ git config --global user.name "Andy Wallace"
2. Enter your Comcast email address:
   $ git config --global user.email Andrew_Wallace@cable.comcast.com
3. Setup your editor:
   $ git config --global core.editor vim
4. Setup you diff tool:
   $ git config --global merge.tool vimdiff

You can run the following command to see all of you Git settings:

		$ git config --list
		core.symlinks=false
		core.autocrlf=true
		color.diff=auto
		color.status=auto
		color.branch=auto
		color.interactive=true
		pack.packsizelimit=2g
		help.format=html
		http.sslcainfo=/bin/curl-ca-bundle.crt
		sendemail.smtpserver=/bin/msmtp.exe
		diff.astextplain.textconv=astextplain
		rebase.autosquash=true
		user.name=Andy Wallace
		user.email=Andrew_Wallace@cable.comcast.com
		core.editor=vim
		merge.tool=gvimdiff
		gui.recentrepo=C:/Users/awalla5075k/1st_Project
		gui.recentrepo=C:/Users/awalla5075k/git-test/CFX_BR_CEMP_CMTools

		awalla5075k@CO183LCETENG08 ~
		$

For more details on initial setup, refer to First-Time Git setup.

SSH Public Key

The EM&M Git Repository System encompasses two servers, a primary server and a fail-over server. This is where EM&M deployments are managed from. In order to gain access to these servers, you'll need to email your public SSH key to the EM&M Configuration Management (CM) team, (Currently this is Andrew Wallace, Robert Sell, and Bruce Woodcock). You can reference Generating Your SSH Public Key for the following instructions.

Execute the following in your new Git Bash window:
$ cd ~/.ssh

- Note the "~/" utilizes the workstation **HOME** variable for your NTLogon.
  If you have a ssh key pair named with your NTLogon, you can probably use that key pair.
  $ ls
  andyw_rsa andyw_rsa.pub config id_rsa known_hosts
  $
  Generate key pair using your NTLogon
  $ ssh-keygen -t rsa -f [Your NTLogon] (all lower-case, no mixed case.)

When emailing your SSH Public Key to the EM&M CM team:

- Your NTLogon key pair name will be the userid used by the gitolite admin tool.
- If you have a need for multiple ssh keys. Contact the EM&M CM team for more information.
- Be sure your git workarea(s) on your windows workstation are on the C Drive.
  - awalla5075k@CO183LCETENG08 /h
  - $ pwd
  - /h
  -
  - awalla5075k@CO183LCETENG08 /h
  - $ cd ~/ (or cd $HOME)
  -
  - awalla5075k@CO183LCETENG08 ~

- $ pwd
- /c/Users/awalla5075k
-
- awalla5075k@CO183LCETENG08 ~
- $

Add SSH Aliases

To reduce typing and minimize ssh key issues, the following is done to provide ssh aliases for the EMM Git System servers by adding a config file under the ~/.ssh on your workstation for your NTLogon.

Edit (or create) ~/.ssh/config and add the following lines adjusted for your NTLogon:

$ vim ~/.ssh/config

```
####################################################################
####################################################################
### EMM Git System SSH Client Config file                      ###
###                                                            ###
### This code block required for EEM Git System Access.        ###
###                                                            ###
### Reference link "EM&M Git Users Guide" in one of the        ###
### following URLs for details.                                ###
###    http://emm-git1.sys.comcast.net  (primary server)       ###
###    http://emm-git2.sys.comcast.net  (fail-over server)     ###
###                                                            ###
### Place this code block in file ~/.ssh/config on your        ###
### workstation. If ~/.ssh/config already exists, add          ###
### this code block to file ~/.ssh/config.                     ###
###                                                            ###
### DISCLAIMER:                                                ###
###    This code block not designed to work with wildcard      ###
###    definition for Host (Host *) in the ~/.ssh/config       ###
###    file.                                                   ###
###                                                            ###
###                                                            ###
####################################################################
Host emm-git primary
        User git
        Hostname emm-git1.sys.comcast.net
        Port 22
        IdentityFile ~/.ssh/[Your NTLogon]

Host emm-gitA fail-over
        User git
        Hostname emm-git2.sys.comcast.net
        Port 22
        IdentityFile ~/.ssh/[Your NTLogon]
####################################################################
####################################################################
```

:wq

This file allows you to enter commands like this:
$ git clone emm-git:foo
Rather than this:
$ git clone ssh://git@emm-git1.sys.comcast.net/foo

Upgrade Git

Follow the same steps for Installing Git in the previous section. Note the expected version release change after you are done.

EM&M Git Related Support

The EM&M Repository system at emm-git1.sys.comcast.net/git/ is a Central Repository supporting build and deployment automation opportunities for EM&M modules. This section focuses on functionality provided for and supported with this system.

GitWeb

emm-git1.sys.comcast.net/git/ is a link to the GitWeb site of the EM&M Central Repository supporting EM&M employees and contractors with configured, central Git repository functionality.

Shared Temporary Repos

For each EM&M Project/Development Group, developers can create temporary repos to back up their workstations. These temporary repos can be used to share code between developers. They can also be used to submit code to be merged with the develop branch. The repo name components for a temporary repository is documented in the following EM&M Git Server section.

EM&M Git System

The emm-git1.sys.comcast.net/git/ provides an immediate backup for EM&M developer workstations. It also provides:

- Full repo workareas based on shared module repos.
- Repo master and develop branches to merge with on the developer workstation.
- Daily Backups and restores to server emm-git2.sys.comcast.net which is physically located in Chicago. The primary server emm-git1.sys.comcast.net is physically located here in Denver at the potomac site.

Access Management

User access is managed with SSH keys. The SSH keys are named with the Comcast NTLogon on EM&M Git System servers. These keys are managed as users in groups and roles.

The group definitions are as follows. Every user (ssh key) is associated with one of these groups:

Groups	Description
@cmadmin	Full privileges on all repos.(This is the EM&M CM Team.)
@developers	Full previleges as creators on tmp repos, RW on @devlead creator directory repos, RW on EM&M repos used for production deployment. EM&M repos are moved from @devlead creator directory repos. No merge to develop or master branches.
@devleads	Have the ability to initialize @devlead creator directory repos and merge to all repo develop branches. No merge to master branch.
@general	Available for assignment to WRITER or READER roles by devleads and developers for repos they own. Enter all users for this EM&M repository server users in this section under appropriate groups. User should be entered in one group only.

Notes:

1. CREATOR will be the userid of the person logged on to the system via a defined ssh key file.
2. This configuration file is currently setup with the following expectation:
   A user is entered once in one of the groups.
3. Group definitions are always lower case. Role definitions are always upper case.

The role definitions are as follows. Users are added to roles by repo owners. Repo owners are usually Development Leads:

Roles	Description
OWNERS	List of users that are assigned push, create, delete, or rewind any branch (less master) or tag in specified repo. Permissions assigned by initial repo owners. Can push to develop branch only if user is in @devlead group. Cannot push to master branch.
WRITERS	List of users that are assigned push only, no rewind access, basically read/write access, by repo owners. Cannot push to develop or master branches.
READERS	List of users that are assigned read only access by repo owners. Cannot push to any branches.

Notes:

1. Initial repo owners for finalized repos are devleads.
2. Role definitions are always upper case. Group definitions are always lower case.

Functional Support

In the following table note that "repo name" is the name of the EM&M Git repository module.

The git command syntax for interacting with the EM&M Git Server is as follows:

- (The following syntax requires you to setup ssh key aliases as instructed above. Reference section Add SSH Aliases.)

Tasks	Git/ssh Suntax	Comments
Clone repository using ssh key Read/Write	$ git clone emm-git:[repo name]	Requires: 1. user ssh key delivered to and implemented by EM&M CM team.

access.		2. User in group with read permissions to "repo name".
Get Read only copy of repo	Go to EMM GitWeb, select desired project, select **snapshot** of desired branch/tag.	This will provide a gzip file that you can install on your workstation.
Create temporary repo for sharing with other developers and lead.	$ git clone emm-git:tmp/CREATOR/[LOB]-[Project]_[Major]_[Minor]	Requires: • User ssh key delivered to and implemented by EM&M CM team. • Dev Lead Area established with Dev Lead who will manage. These temporary repos are also used to submit code for merging with development branches. The repo name components for a temporary repository is as follows: • 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 disk space is needed and temporary repo is not active, it will be removed. • CREATOR - This is substituted with the analyst's logon ID as identified with the analyst's public ssh key name. • LOB - Line of Business, one of CFX,VIDEO,VOICE, or DATA • Project - Name at discretion of Dev Lead. Does not contain / or _. • Major - Number for baseline functionality. • Minor - Number for for partial release to associated Major number. • Dev - Numeric increment for each version of the MAJOR_MINOR release delivered by CEMP Development. • Test - Numeric increment for each version of the MAJOR_MINOR_DEV release delivered by CEMP Development. **These repos are not backed up to the EMM System failover server. Code to be merged with DevLead managed repo.**
Development lead initializes new repo for future production deployment.	$ git clone emm-git:CREATOR/[LOB]-[Project]	Requires: • User ssh key delivered to and implemented by EM&M CM team. • Dev Lead Area established with Dev Lead who will manage. These initial repos are setup with general developer access and ownership by the Development lead. The repo name components for an initial repo setup by a devload are as follows: • CREATOR - This is substituted with the analyst's logon ID as identified with the analyst's public ssh key name. • LOB - Line of Business, one of CFX,VIDEO,VOICE, or DATA • Project - Name at discretion of Dev Lead. Does not contain / or _. • Check you initial code set into the "develop" branch. Before a Development lead initiated repo is merged into the master branch for a production deployment release, it needs to be finalized in the repository by the Configuration Management staff. **Note:** Before code can be QA installed, it must be tagged with format [repo name]_[Major]_[Minor]_[Dev]_[Test] and checked into the "develop" branch. You will need to checkout the newly finalized repo as follows, since the master branch is still empty: $ git clone -b develop emm-git:CREATOR/[LOB]-[Project] **These repos are not backed up to the EMM System failover server. Repo must be Finalized by CM staff.**

Finalize development lead initialized new repo for future QA, Test, and production deployments.	EMM CM Staff updates EM&M Repository and Repository Configurations	Procedure to finalize devlead initiated repos was initially documented in the preamble for devlead created repos of the gitolite.conf file. Refer to section Finalize DevLead Repo of the EMM Git Admin guide for details. Need the following from devleads for this procedure: 1. Repo Devlead FirstName LastName 2. Repo Description 3. Repo Category Verify Development Lead has initiated a an upstream "develop" on the emm-git server for the repo with the following command: $ git push --set-upstream origin develop
Push local repo on your workstation to the EM&M Git server.	$ git push	Requires local repo to have been setup with a "git clone" option that has **write** authority on the EM&M Git server.

- Assign OWNERS, READERS and WRITERS permissions to your repos.
- List OWNERS, READERS and WRITERS for repos you own.
- Enter description for your repo. (Must be owner)
- List your permissions for all repos.
	List remote commands available:
	$ ssh git@emm-git help
	Help for each available remote command:
	$ ssh git@emm-git [remote command] -h
		Users of this EMM Git repository have direct access to the repository server via this defined ssh session 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 tasks listed.
		
		Reference Gitolite, the User's View. This reference provides a great introduction to all functionality available to the users of the EMM Git Repository System. All sections covered include:
		
		1. accessing gitolite
		2. the info command
		3. normal and wild repos
		4. other commands
		• set/get additional permissions for repos you created
		• adding a description to repos you created
		5. "site-local" commands
		6. "personal" branches

Git Support for Eclipse

- EGit **Need to review and construct this section.**

Documentation

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 EM&M deployments and releases review A Successful Git Branching Model by Vincent Driessen in it's entirety. EM&M specifics **will be** covered in this document.

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 repository system, the user must merge their change into the appropriate Main Branch of the targeted module repo. Module repo leads are assigned to aid users with these branching and merging tasks.

Main Branches

The are two Main Branches in any given EM&M repo on the repository system, **develop** and **master**. Supporting branches are used to manage branching and merging tasks performed before and between these Main Branches.
Note:

1. In "Git Speak", **origin** is associated with the EM&M repo on the repository system.

The **develop** branch is the primary integration branch where automatic nightly builds are build from. When the source code in the **develop** branch reaches a stable point and is ready to be released, all of the changes are merged into the **master** branch and then tagged with a release number. 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**.

Feature Branches

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

h7. Creating a feature branch

When starting work on a new feature, branch off from the develop branch.

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

h7. 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 bundled added the feature.

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

Release Branches

May 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.

h7. 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.**

h7. 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

May branch off from: master
Must merge back into: develop and master
Branch naming convention: hotfix-[Major].[Minor].[hf#]
Notes:

1. If there is an active, in-progress release branch, merging should be considered depending on current state of major branches.
2. hf# - 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.

h7. 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(+)
$
```

h7. 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:

ModuleName - The name of the Git repo being released from. The Git repo name is the EM&M Module it supports. See following section on EM&M Modules for how the EM&M Module name is constructed.
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 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

Developer Deployment Branches

The format of the developer deployment branch name is **[ModuleName]_N_N_N_N** where each "n" is a nummeric increment for Major, Minor, Development, QA. 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.

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.

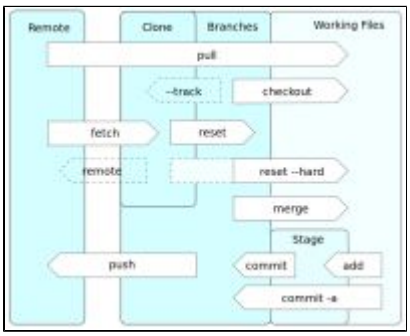
Useful Command Syntax

Syntax	Comments
git diff master origin/master	After "git fetch origin", shows difference between master branch and branch on emm-git repository (origin).
git log --pretty=oneline --graph --decorate	Report representation of current repo. (May need to send to tmp file to see full report - >> tmp.txt)
gitk	Visual representation of current repo.

Cheat Sheets

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

Git Branching/Merging Model
You need flash player installed to preview ppt and pdf files Get ADOBE® FLASH® PLAYER

	vim	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
Git Software
Git Operations
Git reference
Git for Computer Scientists

Understanding Git Conceptually
git fetch and merge, don't pull