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 System has been setup with the following considerations in mind:

1. Encompass and embrace branching and merging to support automation for builds and deployments
2. Production EBF development must always be initiated upon the exact set of source code which was used to generate the problematic version that is in production needing repair
3. The process and standards which we set up:
    a. must be able to give us a consistent and predictable means by which to manage development deployments, QA and production deployments, and production EBF deployments
    b. should minimize the manuals steps and actions that are performed.
    c. must support automated build and deployment of applications
    d. must support the development, operations, cm, and other teams which interact with the system
4. An effort should be made to avoid biasing the processes at the expense of any of these teams

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.

Git is a Distributed Version Control System (DVCS). If you have not worked with Git before, it is recommended you review the Git Reference before moving further in this document. This reference covers the Git "Porcelain" commands that are build on the Git "Plumbing" commands. Reference Git Internals - Plumbing and Porcelain for details.

The Git Reference provides a basis of Git understanding that is assumed for the remainder of this User Guide. The primary reference for the EM&M Git System is git.

The purpose of this document is to:

1. Identify how to set up and maintain a Git environment on and analyst workstations to interface with the EM&M Git System.
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 time files are added to the EM&M Git System, it is by branch and can an entire structure of 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. The Git approach described previously is assumed before updates are made to the EM&M Git System Main Branches.
5. The "develop" and "master" branches are the Main Branches of the EM&M Git System.
6. Where did the term Git originate from? Reference the **History** section of Git (software) in Wikipedia.

Using git is not a "traditional" way of working and is designed to be "peer driven" rather than "hierarchical" like SVN/CVS. In essence, everyone has git commit access, but only locally.

The EM&M Git System stores repos that support release and maintenance of code to development and non-development environments. Access to this system is managed with ssh keys. Those who have access have full git fetch access. They have full git push access with the following limitations:

- develop branch (devleads only – designed for automated builds and deployments to development)
- master branch (Designated for production deployments. The master branch is the default branch if one is not specified)
- v[0-1] tags (Designated for production deployment tagging of verified production installs)

If you setup a git repo in the EM&M Git System, you are the default owner. You can provide and deny access to others who have ssh keys setup in the system. You can also disable and enable git push for repos you are the OWNER of in case you need to freeze the repo for any reason. The EM&M Git System has been setup so developers can share repos prior to any release or configuration management activities. This does not preclude developers from sharing their own git repos with other developers outside of the EM&M Git System.

Standards

This section is maintained to support smooth, stable, and 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.

Benefits of Configuration Management

The Configuration Management (CM) environment provides and maintains an environment for the following benefits. With the EM&M Git System, Branching and merging has become a more integral part of the EM&M CM tool set. Branching and Merging will be leveraged to support build automation and deployment.

- Safeguarding intellectual property—the software assets.
- Helping to improve communication among team members.
- Providing a way to establish clear responsibilities and accountabilities.
- Providing traceability and reproducibility.
- Facilitating reusability.
- Providing for consistency, reliability, and integrity.

Branching and Merging

The ability to merge changes with existing deployments and development sets before an environment is deployed to is essential for build and deployment automation. The environments this system supports must be fully represented in the deployment sets new changes are merged with. Two main branches are maintained in each EM&M Git repos for this purpose, **develop** and **master**.

The **develop** branch is for "automated build" and "automated deployment" of deliverables to development environments. The code sets for this branch should alway be buildable and deployable automatically to the development environment(s). It is up to the development team for the git repo to determine what supporting branches and main branches are needed to determine when the team is ready to "git push" to the "develop" branch. Developers manage feature branches that are contributed to the **develop** branch. The primary distinction between main branches and supporting branches is that main branches are permanently maintained and supporting branches are temporary to the point of being merged with one or more main branches.

The **master** branch is for "automated build" and "on demand deployment" that was attempted and or delivered to the production environment. When a production delivery is verified and confirmed, a production release tag is applied to the master branch node delivered. There are two supporting branches that are utilized between the develop and master branches, release branches and hotfix branches.

The **release** branch is for code that has been developed and tested to the point of being "deployment ready" as determined by the development group. Deployments from this branch are intended for environments outside of the development organization. These environments include, QA, Integration, Stage, and Production.

The **hotfix** branch is for changes that are required for the production environment in a short order. These branches provide for managing quick fixes into production and coordinating updates to the current state of the git repo. For all outstand releases, other hotfixes, and the current develop branch, merges from this branch are required whenever this branch is merged to the **master** branch for a new production deployment of a given git repo.

Branching and Merging Anti-Patterns

The following considerations contributed to the Branching and Merging model described above. Development must continue while all changes to production are coordinated and maintained with development in an organized predictable manner.

- **Merge Paranoia**—avoiding merging at all cost, usually because of a fear of the consequences.
- **Merge Mania**—spending too much time merging software assets instead of developing them.
- **Big Bang Merge**—deferring branch merging to the end of the development effort and attempting to merge all branches simultaneously.
- **Never-Ending Merge**—continuous merging activity because there is always more to merge.
- **Wrong-Way Merge**—merging a software asset version with an earlier version.
- **Branch Mania**—creating many branches for no apparent reason.
- **Cascading Branches**—branching but never merging back to the main line.
- **Mysterious Branches**—branching for no apparent reason.
- **Temporary Branches**—branching for changing reasons, so the branch becomes a permanent temporary workspace.
- **Volatile Branches**—branching with unstable software assets shared by other branches or merged into another branch.
    - **Note:** Branches are volatile most of the time while they exist as independent branches. That is the point of having them. The difference is that you should not share or merge branches while they are in an unstable state.
- **Development Freeze**—stopping all development activities while branching, merging, and building new base lines.
- **Berlin Wall**—using branches to divide the development team members, instead of dividing the work they are performing.

Since git is a Distributed Version Control System (DVCS), each developer maintains their own repo with the corresponding EM&M Git System repo as remote. This allows development to come up with any Branching and Merging approach they deem necessary. The above list should be considered when developing a Branching and Merging approach.

Merge points must be planned and managed to encourage clean usable history and provide commonly known starting points for branches. **It is the responsibility of the developer to understand all merge related issues. Verify with your development lead if you have any questions on merge related issues.**

In order to keep the complexity of branching and merging to a minimum for EM&M Git System managed repos, each repo allows for one **develop** main branch and one **master** main branch. If additional development efforts or production deployments are required, and they can't be handled with the branches defined, new repos are required. Naming and dependency tracking of EM&M Git System repos is required for this and other reasons. It all starts with good comments on each "git commit" so the EM&M gitweb site can be used as a primary reference for the inventory of repos and how they are dependent on each other. Another important reference for each EM&M Git System repo is the CHANGELOG.

References

1. Linus Torvalds recommentation
2. Branching and Merging Primer
3. A Successful Git Branching Model
4. Git Branching and Merging
5. Comparing Workflows
6. Git Branching Model
7. Git Workflows for Successful Deployment

CHANGELOG

In order to maintain EM&M module repo releases for future reference by internal and external audit groups, a CHANGELOG file is maintained for each production deployment. For an example of how a CHANGELOG file is used, refer to URL https://github.com/sitaramc/gitolite/. The following template has been defined for CHANGELOG entries and how they are managed in the EM&M organization. You also get a copy of the CHANGELOG.template via http://emm-git1.sys.comcast.net/git/ from the CFX-EMM-GITSystem repo in the docs directory:

```
YYYY-MM-DD      vM.m            Build ID: [repo]-[branch]-[deployment]-[build#]

                        Description:
                                This template identifies the format of a CHANGE log entry for a
                                single EM&M repo. The text in this Description and component fields
                                defined below should be replaced accordingly for the project this
                                template is being used for.
```

A CHANGE log entry is added to the top of the CHANGELOG file each time a new release of the repo is constructed.

The first line consists of the Date (YYYY-MM-DD), tab, repo release tag (v with Major.Minor increments from previous repo release tag), tab, and Build ID:, uniquely identifies the build that rolled up for this repo release tag.
NOTES:
1. The date is the appropriate Target Release Date (TRD) until the "Go No Go" meeting. It is assigned the date of production deployment when identified in the "Go No Go" meeting.
2. Hotfixes are identified in the repo with a third increment. For example, for repo release tag v.11.2.10, 10 would indicate the 10th hotfix or EBF for release v.11.2. "0" or "null" is not counted.
3. For Build ID:, Unique build id scheme has been proposed to development. Current available alternate is Module release ID tracked in JIRA record summary field. Module ID tag increments are M=Major, m=minor, d=development, t=test.

The components of this change log entry for a given release (i.e., Description, Comcast System Indexes, Latest Production and Development Dependencies, Change Records/Bugs/Defects, Known Issues, Enhancements, New Features) all start with an entry of "None identified". As the release progresses and is prepared, this record is updated by all team members as required. It is verified by CM prior to QA installations. It is reviewed during the EMM release "Go No Go" meeting when identified for production deployment. It is reviewed and completed with the production deployment is completed and verified.

In the event of a hotfix, this record is required. The entries in the component fields of this record need to identify the change for future audit references.

Examples of a CHANGELOGs:
For the gitolite project at github, refer to URL https://github.com/sitaramc/gitolite/.

Refer to the CFX-EMM-GITSystem.git repo at URL http://emm-git1.sys.comcst.net/git/

Dependencies:
Comcast System Indexes:
iTRC Application(s) (Repeat for each iTRC app.)
iTRC devices - Envs (dev, QA, Int, Stage, Prod)
None identified.
None identified - None identified
CADA Host groups - (access)
None identified.

Latest Production Dependencies
Deployable for release YYYYMM-[A|B|OC] or hotfix-YYYYMMDD
None identified.

Latest Development Dependencies
Deployable after release YYYYMM-[A|B|OC] or hotfix-YYYYMMDD
None identified.

Change Records/Bugs/Defects:
None identified.

Known Issues/Plans:
None identified.

Enhancements:
None identified.

```
New Features:
        None identified.
```

NOTE: Reference Git Has Integrity for Git SHA-1 hash details.

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, Integration, Stage, and Production environments. Identify why they are currently needed and what the current mitigation plans are for each variance.

Deployment Rules

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 (git repo) 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.

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 applications on Comcast environments via this DVCS (EM&M Git System).

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, the following higher level directories and file shall be in the 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.
- CHANGELOG - Updated and tagged for each production release tag with a record based on CHANGELOG template entry for each production deployment.

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. Multiple EM&M Git System repo sets can be established to "full module baseline deployments" for a give repo.

NOTE:

- Individual EM&M modules are setup as EM&M Git System repos.

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 committed into the repo with comments as they are adjusted in the environment being deployed to.

NOTES:

1. If the deployment is a Full Module Baseline deployment, the bullet tasks above are not required.
2. Hotfix patch deployments are based on hotfix branches.

No deployments are made to any of the EM&M environments (QA, Integration, Stage, Production) 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 elements 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 **deployment 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.
3. The build system supporting the "develop" branch is managed by the development team. Inputs and updates to this system come from the development staff with deployment/installation instructions for the CM and EPA teams to review and follow.

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)

NOTE: This Git installation is recommended by the Git SCM site for Windows Installations. Since the EM&M Analyst primary workstation is Windows 7, this is the prefered approach using openSSH. Another option is An Illustrated Guide to Git on Windows. Both provide the same tools. The main difference is the second option uses putty instead of openSSH. In the Putty case, the contents of environment variable SSH_GIT is set to use plink.exe. This will not work for the openSSH option.

Initial Git Configurations

After you have installed Git successfully, you have full Git functionality on your workstation. In order to interface with the EM&M Git System, the following needs to be performed:

- Identify or Set $HOME variable on your (Windows 7) Workstation.
- Build and Configure SSH key Usage.
- Identify yourself in Git:
    - Name
    - email
    - Setup your default editor
    - Setup the diff tool you will use
- Add SSH Aliases.
- Verify Git remote "origin".

The above bullets are covered in following sections with Links to each maintained in the "Table of Contents".

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

Identify the $HOME variable on 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. Verify there is a **HOME** variable in the **System variables** list. This variable is referenced by SSH when supporting remote functionality with the EM&M Git System.
    a. Should be set to C:\Users\ [your NTLogon].
    b. Create (New button) or adjust (edit button) the **HOME** system variable accordingly.

Build and Configure SSH key Usage

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 more details on the following instructions.

Execute the following in your new Git Bash window.
$ cd ~/.ssh (c:\Users\awalla5075k\.ssh)

- If your do not have a ~/.ssh directory, create it:
  - $ mkdir c:\Users\awalla5075k\.ssh (Substitute your ntlogon for awalla5075k)
  - Set the ~/.ssh permissions to 755
  - Set the file permissions in ~/.ssh to 744
- Note the "~/" utilizes the workstation system **HOME** variable for your NTLogon.
  Generate key pair using your NTLogon
  $ ssh-keygen -t rsa -f [Your NTLogon] (all lower-case, no mixed case.)
- **NOTE:** It is recommended you enter nothing for the pass phrase.

The following two files will be generated:

- [Your NTLogon] - Your **private** key file
- [Your NTLogon].pub - Your **public** key file
  NOTE: Never send your private key in an email.

Email your **public** key file to the CM Team (Robert Sell, Andrew Wallace, Bruce Woodcock, Abhilash Nair). Reference the Access Management section of this document. Select one of the access mentioned indicating this in the email. CC your manager on this email.

You will not be able to interact with the EM&M managed remote repos until your ssh key has been setup by the CM team.
In the meantime, you can create and experiment with Git repos using the "git init" command.

Once the EM&M CM team has replied back to you about successfully setting up your ssh public key on the EM&M Git System, you are ready to move to the Add SSH Alias section below.

Identify Yourself in Git

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 ~
		$

- 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. Add a config file under the ~/.ssh on your workstation for your NTLogon as follows.

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

Enter the following command in your git bash window to confirm your ssh key is functioning correctly. The output identifies the repos you can work with on the EM&M Git System:
$ ssh git@emm-git info

Verify Git remote "origin"

For a full set of supported options now available to you, refer to the Functional Support section of this document. This section explains how to setup git repos on the EM&M Git System that you can populate with file and code elements.

One you have established a git repo local on your workstation from the EM&M Git System, you can verify the git remote origin in the local copy on your workstation as follows. The git remote "origin" should be setup for communication between your workstation repo and the associated EM&M Git System repo. Reference git remote for more details.
$ cd ~/path-to-repo
$ git remote -v show origin

First "git push" to The EM&M Git System

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

- Are you familiar with the Git Reference site?
- Are you familiar with the "Git Approach"? see **Notes:** 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 develop on a feature branch and merge your changes to your development team’s supporting branches on the EM&M Git System?
- If your development team has automated build, test, and deployment for “git push” to the EM&M Git System repo develop branch:
    - 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 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.

NOTES:

1. For any git command, you can enter git [command] –h for command documentation.
2. EM&M Git System repo branch and tag access rules are configured and maintained in the gitolite-admin repo, conf/gitolite.conf file.
3. The Git Reference site is a great quick reference for git "Porcelain" commands. Reference Git Internals - Plumbing and Porcelain for details.

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 **and only one** of these groups. Note the @devlead group is the only group (other than @cmadmin) that has "git push" functionality for the "develop" branch.

Groups	Description
@cmadmin	Full privileges on all repos.(This is the EM&M CM Team.)
@developers	Full privileges as creators on tmp repos. No push/merge to develop or master branches. Available for assignment to roles by repo owners/creators.
@devleads	Full privileges as creators on tmp repos. Have the ability to initialize @devlead creator directory repos and merge to all repo develop branches. No merge to master branch. Available for assignment to roles by repo owners/creators.
@general	Full privileges as creators on tmp repos. No push/merge to develop or master branches. Available for assignment to roles by repo owners/creators. This group intended for non-developers.

Notes:

1. CREATOR will be the userid of the person logged on to the system via a defined ssh key file.
2. The gitolite configuration file is currently setup with the following assumptions:
    a. **A user is entered once in one of the groups.**
    b. If a user is in a group, they are available for assignment by repo owners to all available Roles (OWNERS, WRITERS, READERS, MSLEADS, MSDEVS).
    c. All users are restricted from "git push" on EM&M Git System repos hotfix, release, develop, and master branches. Users in the Devlead group can "git push" to EM&M Git System develop branches.
    d. Group definitions are always lower case. Role definitions are always upper case.
3. Groups define baseline privileges, roles enhance the baseline privileges for a given user, on a given repo, by the repo owner/creator.

The role definitions are as follows. Users are managed to roles by repo owners at the repo level. Note that Git branch and tag capabilities for a given user are set according to the group they are in:

Roles	Description
OWNERS	List of users that have git push, create, delete, or rewind branch or tag in specified repo. Permissions assigned by repo owners. Repo creators are initial repo owners.
WRITERS	List of users that are assigned push only, no rewind access, basically read/write access, by repo owners for given repo.
READERS	List of users that are assigned read only access by repo owners.
MSLEADS	Managed Service Leads. Same as OWNERS role.
MSDEVS	Managed Service Developers. Same as WRITERS role.

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

The EM&M Configuration Management (CM) team provides support for the EM&M Git System described 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 Syntax	Comments
Copy an EM&M Git System repo to your workstation with workfiles set to the head of a given branch.	$ git clone -b [branch name] emm-git:[repo name]	When working with a git remote system, "git clone" is used to establish a Git repo on your workstation that is managed on the EM&M Git System. This syntax requires the following: 1. user ssh key delivered to and implemented by EM&M CM team. 2. The Add SSH Aliases section of this document has been followed. 3. User in group or ROLE with read permissions to "repo name". 4. Valid branch name for the repo. Refer to EM&M Git GitWeb summary for repo in question, Heads section for branch names.
Get Read only snapshot copy of EM&M Git System repo via Web	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]
	$ cd [LOB]-[Project]_[Major]_[Minor]
	$ git checkout -b [branch name]
	$ *Add your files, git add and commit ...*
	$ git push --set-upstream origin [branch name]
		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. Requires:
		
		• User ssh key delivered to and implemented by EM&M CM team.
		• Dev Lead awareness who will add in coordinating with team.
		
		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.
		
		Notes:
		
		1. Initially, the default master branch on the new tmp repo is not valid until a file is committed to it.
		2. You will not be able to "git push" to the remote **master, hotfix, release, or develop** branches.
		3. You must be a developer lead to "git push" to the remote **develop** branch.
		4. You will not be able to create a tag that starts with v[0-9]. This is reserved for Production Releases.
		
		These repos are not backed up to the EMM System failover server. Code to be merged with DevLead managed repo.

Development lead initializes new EM&M Git repo for future production deployment.	$ git clone emm-git:CREATOR/[LOB]-[Project]	This will create a repo in the EM&M Git System and create a copy of it on the executors workstation. It will need to be loaded on your workstation using Git add, commit. Then use Git push to update the corresponding repo in the EM&M Git System. An example command sequence is listed below. 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. **Notes:** 1. Initially, the default master branch on the new tmp repo is not valid until a file is committed to it. 2. You will not be able to "git push" to the remote **master, hotfix, release, or develop** branches. 3. You will not be able to create a tag that starts with v[0-9]. This is reserved for Production Releases. 4. 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] Command sequence example: (Local on your workstation where the repo will be created)

		[userid]@C0183xxxxxxxx/c/GIT_stuff
		$ git clone emm-git:[userid]/[repo name]
		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.
		[userid]@C0183xxxxxxxx/c/GIT_stuff
		$ cd [repo name]
		[userid]@C0183xxxxxxxx/c/GIT_stuff/[repo name]
		$ git checkout -b develop
		Switched to a new branch 'develop'
		[userid]@C0183xxxxxxxx/c/GIT_stuff/[repo name]
		(Add/move files to this directory. Use "git status" as needed)
		$ git add .
		[userid]@C0183xxxxxxxx/c/GIT_stuff/[repo name]
		$ 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 ...
		
		[userid\]@C0183xxxxxxxx/c/GIT_stuff/[repo name]
		$ git push --set-upstream origin develop
		
		These repos are not backed up to the EMM System failover server. Repo must be Finalized by CM staff.

Finalize development initiated 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 the upstream "develop" branch on the emm-git server for the repo with the following command: $ git push --set-upstream origin develop **All users of pre-Finalized repos should re-clone their local copy of the repo from the finalized location in the EMM Git System.**
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.

- Manage users to available roles for repos you own.
- Maintain/modify description for your repos.
- List your permissions for repos you have access to.
	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.
		As a repo owner/creator, you can manage users of a given repo to the following roles defined earlier in this guide:
		
		• OWNERS
		• WRITERS
		• READERS
		• MSLEADS - Managed Service Leads
		• MSDEVS - Managed Service Developers
		
		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 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.

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. 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 by development 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.

The are two Main Branches in any given EM&M git repo on the repository system, **develop** and **master**. These branches are for deliverables that build automatically. 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. 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 **release** branch. 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.

h7. 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'
$
```

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 together 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

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.

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

Should 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 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

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.

- [ModuleName]_N_N_N_N

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.**

The format of the developer deployment branch name is **[ModuleName]_N_N_N_N** where each "N" is a nummeric 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.

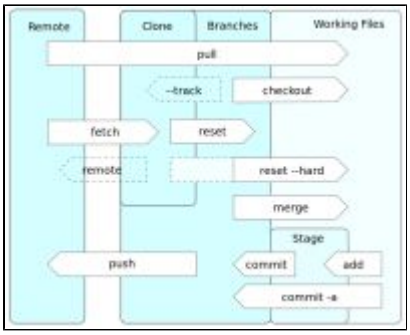
Useful Command Syntax

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 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
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