Version Management Implementation

Table of Contents

- EMM Git Overview
  - EM&M General Git Approach
  - Git Compared to CVS
- EM&M Git System Access
  - Access Management
- Standards
  - Benefits of Configuration Management
  - Branching and Merging
    - Branching and Merging Anti-Patterns
    - References
  - CHANGELOG
    - Dependencies
    - Module Deployment Policy
- GitWeb
- Shared Temporary Repos
- EM&M Git System

EMM Git Overview

The EM&M Git System is the EMM Version Management Repository. It is based on the Git tool. Refer to About Git for more information on the Git tool. It is managed with the Gitolite repository hosting 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. This reference covers the Git "Porcelain" commands that are build on the Git "Plumbing" commands. Reference Git Internals - Plumbing and Porcelain for details. Review The Three States for a basic understanding of your git working environment.

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

EM&M General Git Approach

Each time files are added to the EM&M Git System, it is by branch and a structure of files.

1. The **Git approach**: *(Locally on your DVCS Git repo)*
    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.
    e. Final testing will include Build and Deployment automation in the final development environment.

NOTES:

1. The Git approach described previously is assumed "completed by development" before updates are made to the EM&M Git System Main Branches.
2. The **develop** and **master** branches are the Main Branches of the EM&M Git System.
3. Git Has Integrity for Git SHA-1 hash details.
4. 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.

Git Compared to CVS

The primary differences between the EM&M Git System and EM&M CVS Repository are:

- Git's primary component is a snapshot of a structure, for CVS, it's a file.
- CVS Modules often overlap for shared or common code, Git repos do not. This will make Git repo dependencies clearer.
- We are managing two branches for each Git repo rather one for each module in the CVS Repository.

Using the EM&M CVS Repository, branching and merging was avoided due to the CVS implementation of branching and merging. It is complex for a file oriented system. It is also disk and network intensive due to large amounts of required copying of files to represent branches.

As you use Git, these differences will be more apparent. To get started, review Git Basics, it's only one page. Develop a clear understanding of what a Git snapshot is.

Be sure to review The Three States your work files reside in. This provides a basic understanding of your local git repo working environment.

In CVS, the following procedure was followed to promote and manage changes to the EM&M production environments using EM&M CVS Repository. It was based on one code set that can be thought of as a single branch. Individual file revisions where tagged and used to establish code sets as deployments where identified, tested, and released.

1. Establish a local work area of a CVS module from the CVS Repository to work in.
2. Modify files in local work area for current task.
3. Update local work area from the CVS Repository identifying all changes and required merging changes.
4. Coordinate your local work area with changes found from previous update.
5. Check-in changes from your local work area to the CVS repository establishing a new HEAD set of file revisions.
6. Tag all current file HEADS in the CVS Repository for the module you're changing.
7. Notify EM&M CM team of module new tag set ready for QA deployment.
8. QA team requests QA deployment of new module tag set.
9. CM team creates read-only work area based on new tag set for module.
10. CM team deploys to QA environment.
11. EPA team deploys to Integration, Stage, and/or Pre-prod at PM request.
12. This entire procedure repeats until QA Certifies or the work is cancelled.
13. Hold GoNoGo to identify release sets of certified modules.
14. CM Team packages release sets.
15. EPA team deploys release sets to production based on GoNoGo schedule.

The EM&M Deployment Workflow managing the previous procedure using CVS will not change as we substitute "using Git" initially. It will probably morph some as build and deployment automation begins to take shape.

Following is the procedure to to promote and manage changes to the EM&M production environments using EM&M Git System:

1. Establish a local Git repo from the EM&M Git System to work in.
2. Create a branch to commit your changes to.
    a. Likely based on latest **develop** work if it's new functionality that will or is scheduled for production deployment.
    b. Is based on latest production (from the **master** branch) if it's a correction to the production deployment.
    c. Is based on the head of a **release*** branch if correcting an issue found in one or more QA, Integration, or Staging environments.
    d. Is based on the head of a **hotfix*** branch if correcting a production issue.
3. Commit deltas to your branch while you develop.
4. Once the code set in your branch is stable:
    a. Update the branch in your repo (**develop** or **master**) to the head of the corresponding main branch in the EM&M Git System.
        i. If you are updating to an established **release*** or **hotfix*** branch, your will need to merge and commit to the appropriate main branch as well.
    b. Merge to the local branch in your repo that you just updated.
    c. Resolve any conflicts.
    d. Commit to the local branch. (matches managed branch in EM&M Git System repo)
    e. Do one of the following dependent on current process:
        i. Submit to devlead to push to EM&M Git System repo **develop** branch.
        ii. Submit to EM&M CM for push to EM&M Git System repo **release** or **hotfix** branch.
5. At this point depending on type of update:
    a. CM Team makes new **release*** available for QA Install based on development completion of:
        i. Successful merge and commit to **develop** branch.
        ii. Successful build and deploy in latest development environment for repo.
    b. CM Team makes updated **release*** available for QA Install based on same success criteria described above.
    c. CM Team makes new *hotfix* available for QA Install based on same success criteria described above.
    d. CM Team makes updated *hotfix* available for QA Install based on same success criteria described above.
6. QA Team Requests new or updated **release*** or *hotfix* for QA Deployment. If not yet automated:
    a. CM Team deploys to QA.
7. Operations team deploys to Integration, Stage, and/or Pre-prod at PM request.
8. Procedure repeats until QA Certifies new or updated **release*** or *hotfix*.
9. Hold GoNoGo to identify release set of QA Certified repos.
10. CM team makes Certified repos available for production deployment. If not yet automated:
    a. CM Team packages Certified repos for Operations Team.
11. Operations Team updates for deployment and verification of production deployment of Certified repo.

The procedure using the EM&M Git System allows for management of new deployments, release fixes, and production fixes with the same procedure steps. It also provides for development success with branch merging, builds, and deployments. All development controlled changes occur prior to any deployments to QA, Integration, and Production environments. This reduces churn being experienced with initial builds and deployments to the QA environment.

Deployments are more predictable and less labor intensive if development deployments are successful in a development controlled environment before any official deployments are done.

EM&M Git System Access

The EM&M Git System stores repos that support release and maintenance of code to development and non-development environments. Users with a valid Comcast ID have read access to repos in this system. Modify/Update Access to this system is managed with gitolite. Modify/Update Access to the EM&M Git System has the following role configurations:

- develop branch (**devleads only** – designed for automated builds and deployments to development)
- release branches (**CM only** - Designated for initial deployments to QA environment. Removed after all required branch merges and deployments completed.)
- hotfix branches (**CM only** - Designated for production emergency bug fixes. Removed after all required branch merges and deployments completed.)
- master branch (**CM only** - Designated for production deployments. The master branch is the default branch if one is not specified)
- v[0-1] tags (**CM only** - Designated for production deployment tagging of verified production installs)
  Note: As a user of the EM&M Git System, your local git repo will have full git functionality of the above branches and tags. You will be able to read them from the EM&M Git System. You will not be able to modify them on the EM&M Git System corresponding repos.

If you setup a EM&M 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. This functionality is provided via ssh sessions managed with Gitolite. For details refer to EMM Git System Supported Functionality and Gitolite, the user's view.

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.

Access Management

User access is managed with Gitolite and 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 and application userids configured for build and deployment automation procedures and tasks.

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.
3. For a given repo, a user should be associated with only one ROLE.

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. Version Management is a primary component of CM Management. 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
- Promote accountability.
- Providing trace ability.
- Promote reusable code.
- Facilitating re-usability.
- 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 to the EMM Deployment Workflow 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 a **Development Lead controlled** environment. The goal is to have the code sets for this branch be **automatically** buildable and deployable to the development environment. The head of the **develop** branch is used to create release branches for QA, Integration, Staging, and Production environments. It is assumed to be buildable and deployable if it is in the **develop** branch. No development changes should be **committed** to the **develop** branch until they have been **merged** with the **develop** branch. The **develop** branch is only merged to from other branches where change and development occurs.

The development team responsible for a given git repo determines what supporting branches and main branches are needed to determine when the team is ready to "git push" to the **develop** branch maintained on the EM&M Git System repo. Developers manage feature branches that contribute to the **develop** branch. The primary distinction between main branches and supporting branches is that main branches are permanently maintained in the EMM Git System 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 is 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. All currently outstanding branches for releases, other hotfixes, and the current develop branch, must be merged to from a given **hotfix** branch. This is required whenever a **hotfix** branch is merged to the **master** branch for a production deployment.

The EM&M Git System provides and supports three Role Based Codelines. The **master** branch **head** contains the most recent production deployment. The **develop** branch **head** represents the latest development completed work scheduled for production deployment, and **feature** branches are for isolated development work in progress not yet scheduled for production deployment.

The changes or deltas merged to the EM&M Git System main branches (develop, master) can and will vary in content. The initial focus is on the resulting commit after the merge verifying the following:

- develop branch - Build and deploy to development is automated and works in a functioning development controlled environment.
- master branch - Build and scheduled deploy to production is automated and works in a functioning EMM (Dev,CM,Ops) controlled environment.
    - Note: Before going to the master branch, the development auto build and deployment is further reviewed and tested in one or more of the following environments via the **release*** and **hotfix*** branches:
        - Quality Assurance (QA)
        - Integration
        - Stage

Refer to the EM&M Branching and Merging Workflow for a high level view of the EM&M Branching and Merging approach.

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 the main 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. Git Branching and Merging
2. Git for Computer Scientists
3. Linus Torvalds recommentation
4. Branching and Merging Primer
5. A Successful Git Branching Model
6. Git Branching and Merging
7. Comparing Workflows
8. Git Branching Model
9. 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 EMM Git repo. A unique version of the CHANGELOG file is associated and tracked for each repo branch via the branch value of the Build ID. The following Build ID components are used to identify Artifactory maintained objects and executables for the associated repo branch head:

- [repo] - Module name is the name of the repo
- [deployment] - The Major and Minor numbers of the release
- [build#] - The automatically generated number of the build

For an example of how a CHANGELOG file is used, refer to CHANGELOG file stored with the EMM Git System tree. The following template has been defined for CHANGELOG entries and how they are managed in the EM&M organization:

```
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. Build ID: Components defined below. Used to track deployment status via branch and identification of deployables in all EM&M deployment systems and environments.
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.

All Dependencies MUST be identified to support automated build and deployment procedures. Refer to "Dependencies:" section of this description and the Dependencies section of the *Version Management Implementation* document for details.

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 device FQDN - Envs (dev, QA, Int, Stage, Prod)
None identified.
None identified - None identified
CADA Host group(s) Name(s) - (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.

Other Dependency
None identified.

Change Records/Bugs/Defects:
None identified.

Known Issues/Plans:
None identified.

Enhancements:
None identified.

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

NOTES:
#1st line composed of date/time stamp, production release tag, and Build ID.

1. a. Production Release Tag v[M,m] - Repo deployment – M.m – Major.minor number of repo deployment
   b. Build ID =[repo]-[branch]-[deployment]-[build#]
      i. repo – EM&M Git Repo name
      ii. branch – EM&M Git Repo Branch type, on of:
         1. F – feature branch.
         2. D – develop branch
         3. R – release branch
         4. H – hotfix branch
         5. M – master branch
      iii. deployment – M.m – Major.minor number of repo deployment
      iv. build# - automatically generated build increment.
2. Links for Comcast dependent systems:
   a. iTRC - Information Technology Resource Center
   b. Production CADA - Comcast Authentication and Delegate Access System
   c. Non-Production CADA - ULA CADA System
   d. NSG - NE&TO Stats Grapher
3. Latest Development and Production dependencies are for correct deployment order and identifying all deployment dependencies that need to be installed prior to this deployment. Identifies previous deployment(s) this deployment must follow for each Main Branch (develop, master).
   a. Scheduled EM&M releases are "A" and "B" for each month.
   b. Un-scheduled EM&M releases are Off Cycle (OC) or Hotfix for Production bugs (also known as Emergency Bug Fixes (EBF))
4. Copies of the CHANGELOG.template (above) or CHANGELOG-blank-record.txt via http://emm-git1.sys.comcast.net/git/ from the CFX-EMM-GITSystem repo in the docs directory.

Dependencies

Dependencies are listed in the CHANGELOG. Specific dependencies like iTRC device FQDNs and CADA Host Group names which are links in the respective systems are required.
NOTE: There are two CADA instances, CADA and ULA CADA.

The iTRC device FQDN links and CADA Host Group links are provided for build and deployment automation requirements. All know "additional" dependencies should be listed under "Other Dependency", one record at a time using the format indicated. The following list is maintained for "Other Dependency" considerations.

- 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.

Module Deployment Policy

Deployments are **Full Module Baselines** fully replacing any current installation of the module. This ensures a single dependency on the current installation of the module and verification that there is a full handle on the deployment requirements for the module. It also allows for build and deployment automation. Deployment variances found during the deployment process need to be captured and corrected to enable this policy.

Hotfix branches are production patches that must be merged into the main branches to ensure correct content for future deployments. Hotfixes branches are only managed for current production deployments. This is the only place in the EMM Branching and Merging Workflow where patches are planned for.

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.

The deployment of a module for a given environment is identified with module name (repo name) and the release identifier (Tag containing

Major.minor). This allows for testing and comparison activities on the given environment. Multiple EM&M Git System repo sets can be established for "full module baseline deployments" for a give repo.

Directory and file permissions for unix "world" access shall be "read-only". The recommended unix file permission setting is:
$ chmod 754

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, the following bullets must be addressed.

- All elements of the target deployment **MUST** be identified and scripted for each target environment deployment.
- All **variance** between the repo elements for the release and deployed elements in the environment are identified and cataloged.
- All identified variances from the full module baseline are identified in the repo with README file comments as they are adjusted in the environment being deployed to.

If the deployment is a Full Module Baseline deployment, the bullet tasks above are not required.

Hotfix patch production deployments are based on hotfix branches. These branches are merged into the **master** and **develop** branches to support future Full Module Baseline deployments. Current outstanding **release*** and **hotfix*** branches must also be merged to appropriately.

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 retrievable from Git using the module version tag. The deployment set is a full representation of the Module release being deployed. This includes all elements that changes can be made to for the module in question.
    a. Conditionally, source code files in the repo are tagged with the release tag they contribute to. Resulting executables from source file set builds are deployed to the environment.
    b. The deployment is done under a structure that identifies the repo release elements that are not in the repo since they are objects from the build that used 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. Deployment variances cause environment deployment failures.
3. Environment deployment failures are listed in the CHANGELOG until they are corrected by development.
4. 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 Operations teams to review and follow.

GitWeb

EM&M GitWeb Repo Server is the link to the GitWeb site of the EM&M Git System. It supports EM&M employees and contractors with Git repo reporting.

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.
- Regular Mirroring to the EMM Git System failover server which is physically located in Chicago. The primary server is physically located here in Denver at the potomac site.