
Overview

Overview

This guide presents git usage per the GitFork workflow.

Tips



- [Forking Workflow](#)¹ - Substitute **GitHub** for **Bitbucket** when reading this. Good explanation of this workflow.
- [Forking Projects](#)² a good introduction to GitHub forking.
- [Fork and Pull Workflow for Git Beginners](#)³

Company repositories (repos) are usually managed using the Enterprise [GitHub](#)⁴ system. It is based on the [Git](#)⁵ tool. Refer to [About Git](#)⁶ for more information on the Git tool.

GitHub should be configured for company usage to use standard GitHub functionality to support collaboration. Specifically the GitHub access for organizations and repos, [GitHub fork](#)⁷ and [pull request](#)⁸ functionality.



The above links point to general open-source GitHub documentation. Company GitHub Enterprise is accessed with company specific access management. **Do not use company generated SSH keys to access the general open-source GitHub system.**

(Introduce MDP process/approach here)

¹ <https://www.atlassian.com/git/tutorials/comparing-workflows/forking-workflow>

² <https://guides.github.com/activities/forking/>

³ <https://reflectoring.io/github-fork-and-pull/>

⁴ <https://github.com/>

⁵ [http://en.wikipedia.org/wiki/Git_\(software\)](http://en.wikipedia.org/wiki/Git_(software))

⁶ <http://git-scm.com/about>

⁷ <https://help.github.com/articles/fork-a-repo/>

⁸ <https://help.github.com/articles/creating-a-pull-request/>

1. New to Git?

If you are new to Git, watch the videos at [Git Documentation](#)⁹. The time required is less than 30 minutes. It's a great investment of your time.

The GitHub repos and access are configured to use Github fork and Pull functionality. GitHub pull requests are required for the main branches.

This rest of this overview section covers the following:

- Identifies the **Prerequisites** that need to be addressed.
- Provides **Notes** on issues that have caused questions as this system has been rolled out and used.
- Outlines a typical **General Session** you are likely to encounter as you use and interact with the EM&M Git System.

2. Prerequisites

You will need to install git to your workstation. Reference [git-scm](#)¹⁰ for details.

2.1. Notes

1. Identify Module/Repo naming standards for your team
2. For the GitFlow workflow, you will need Write permission to be able to `git push` to the remote (GitHub repo) **master**, **hotfix**, **release**, or **develop** branches. If you incorporate GitHub fork functionality, you can use GitHub "pull request" functionality with your team.
3. Code reviews are required for all merges to develop and master branches. This can be done with the **New pull request** button on the GitHub repository home page.
4. For any git command, you can enter `git [command] -h` for command specific documentation.
5. Can't *git checkout* to a new or different branch? Reference [Stashing and Cleaning](#)¹¹.

⁹ <https://www.git-scm.com/doc>

¹⁰ <https://git-scm.com/>

¹¹ <http://git-scm.com/book/en/v2/Git-Tools-Stashing-and-Cleaning>

6. Use **gitk** on your window git bash session for a good GUI of your current **local** repo branching and merging picture.

3. General Session

As development and unit testing occurs, most interaction with git will be through your local git repo on an isolated branch. You will interact with the GitHub when:

1. Setting up a new repo in GitHub.
2. Forking a repo on the GitHub interface prior to using "git clone" to copy to your workstation.
 - a. Be sure to add an additional "git remote" to your cloned repo for the repo you forked from.
3. Write permissions on the repo your forked from will be limited.
4. Using the git pull and push commands to your GitHub forked repos and repos you forked from.

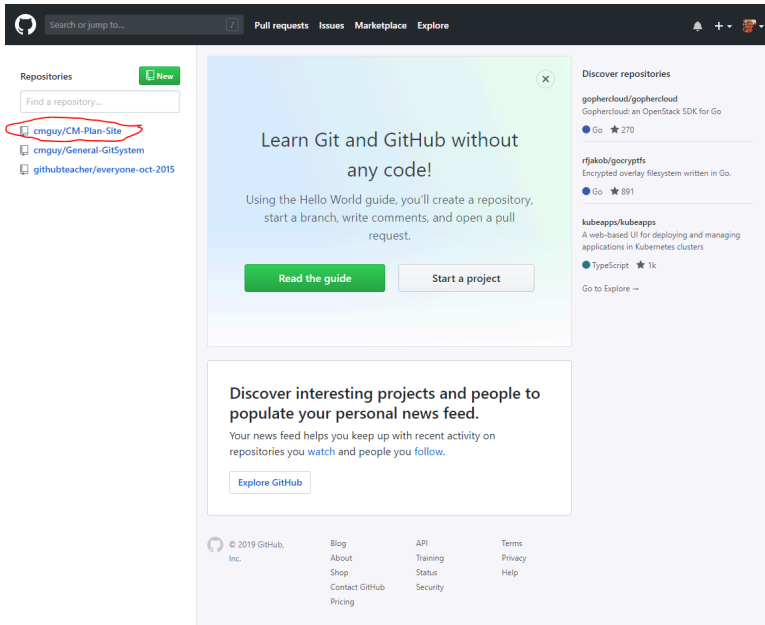
Before you do any `git commit` on your local git repo, you need to:

1. Fork the repo to be worked on to your GitHub account
2. Establish a workstation local repo (`git clone` from your GitHub forked repo)
3. Establish an additional git remote in your workstation local repo for the repo you forked from.
4. Identify available branches on local repo (`git branch -a`)
5. Establish working directory with appropriate branch head (`git checkout [branch name]`)
6. Create new branch to code on (`git checkout -b [new branch]`)
7. Update and/or establish .gitignore file. It Specifies intentionally untracked files to ignore.

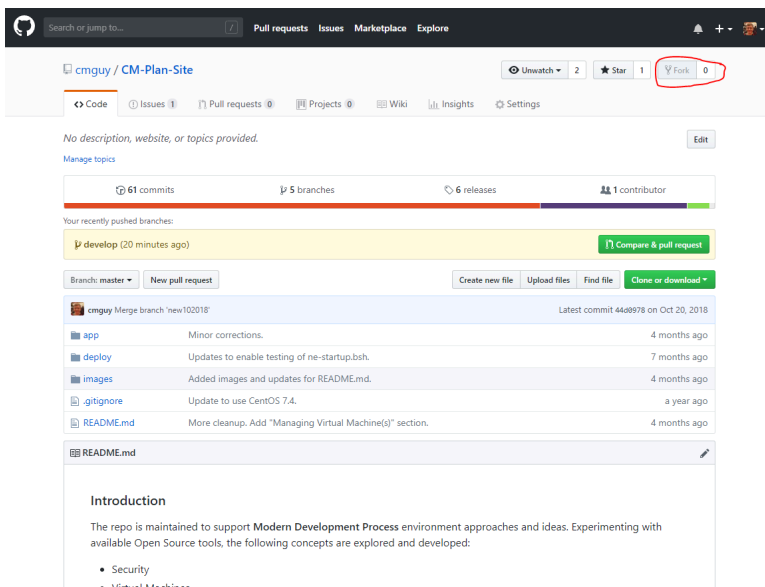
4. GitHub and Git Interfaces

When you followed the Git Installation Procedure previously for your workstation, the recommended Git Bash Command line Interface (CLI) and the Git Graphical User Interface (GUI) for your workstation were both installed.

Before using git locally on your system, you'll need to logon to your GitHub profile and select a repo:



You can Fork the selected repo into your github account:



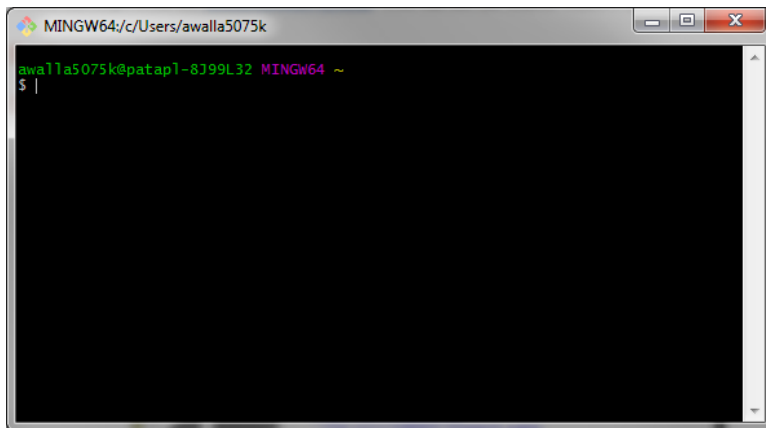
4.1. Delivery

Now you have a profile repo forked to your local profile. You will deliver your updates to this repo after you worked and tested them on the workstation repo you cloned from this repo. If you have Write permissions on the organization repo, you can clone directly from the organization repo and deliver there.

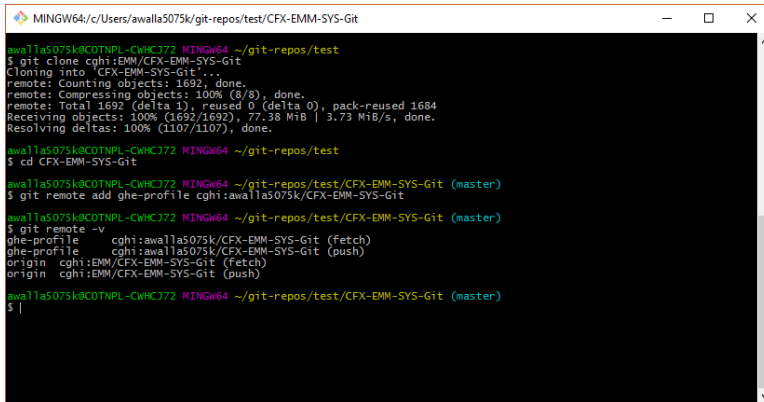
4.2. Using Git Bash:

It is helpful to understand several Git **command line** options before using the Git GUI. Following are examples of initial Git functionality using both the Git Bash and Git GUI tools. Enter "git help" on the git bash command line for an introduction to git commands you will be using. Following are a few examples to get you started.

Establish a Local Git cloned Repo, add remote for the GitHub Enterprise (ghe) profile repo you established previously:



- Clone Existing Repository in a directory you establish for Git repos using **git clone**. Add additional remote for your GitHub profile copy of the repo. At this point you are ready to prepare updates and share your work in collaboration with your team.



```

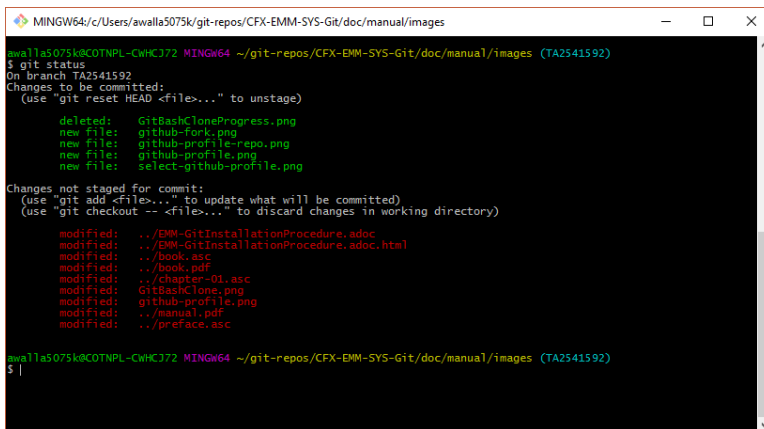
MINGW64: c:/Users/awalla5075k/git-repos/test/CFX-EMM-SYS-Git
awalla5075k@COTNPL-CWHC372 MINGW64 ~/git-repos/test
$ git clone cghi:EMM/CFX-EMM-SYS-Git
Cloning into 'CFX-EMM-SYS-Git'...
remote: Counting objects: 1692, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 1692 (delta 1), reused 0 (delta 0), pack-reused 1684
Receiving objects: 100% (1692/1692), 77.38 MiB | 3.73 MiB/s, done.
Resolving deltas: 100% (1107/1107), done.
awalla5075k@COTNPL-CWHC372 MINGW64 ~/git-repos/test
$ cd CFX-EMM-SYS-Git
awalla5075k@COTNPL-CWHC372 MINGW64 ~/git-repos/test/CFX-EMM-SYS-Git (master)
$ git remote add ghe-profile cghi:awalla5075k/CFX-EMM-SYS-Git
awalla5075k@COTNPL-CWHC372 MINGW64 ~/git-repos/test/CFX-EMM-SYS-Git (master)
$ git remote -v
ghe-profile      cghi:awalla5075k/CFX-EMM-SYS-Git (fetch)
ghe-profile      cghi:awalla5075k/CFX-EMM-SYS-Git (push)
origin          cghi:EMM/CFX-EMM-SYS-Git (fetch)
origin          cghi:EMM/CFX-EMM-SYS-Git (push)
awalla5075k@COTNPL-CWHC372 MINGW64 ~/git-repos/test/CFX-EMM-SYS-Git (master)
$ |

```

- The **git status** command provides current status anytime you need it. It also recommends the next likely git commands you will use based on your current status.



You must be local (In the directory) of a Git repo for **git status** to work correctly.



```

MINGW64: c:/Users/awalla5075k/git-repos/CFX-EMM-SYS-Git/doc/manual/images
awalla5075k@COTNPL-CWHC372 MINGW64 ~/git-repos/CFX-EMM-SYS-Git/doc/manual/images (TA2541592)
$ git status
On branch TA2541592
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    GitBashCloneProgress.png
    new file:   github-fork.png
    new file:   github-profile-repo.png
    new file:   github-profile.png
    new file:   select-github-profile.png

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   ../EMM-GitInstallationProcedure.adoc
    modified:   ../EMM-GitInstallationProcedure.adoc.html
    modified:   ../book.asc
    modified:   ../book.pdf
    modified:   ../chapter-01.asc
    modified:   GitBashClone.png
    modified:   github-profile.png
    modified:   ../manual.pdf
    modified:   ../preface.asc

awalla5075k@COTNPL-CWHC372 MINGW64 ~/git-repos/CFX-EMM-SYS-Git/doc/manual/images (TA2541592)
$ |

```

Checkout a New Branch

- To identify and select a branch to work on:
 - Identify available branches - **git branch -a**
 - Select available branch to branch from - **git checkout [any available branch]**

- Create new branch based on selected branch **git checkout -b [new branch name]**



1. There is a * by the current local branch that is active
2. Local branches are listed with branch name only

```
MINGW64/c/Users/awalla5075k/git-repos/test/CFX-EMM-SYS-Git
jwalla5075k@COTNPL-CWHC372 MINGW64 ~/git-repos/test/CFX-EMM-SYS-Git (master)
$ git checkout -b US1234567/CFX-EMM-SYS-Git_2_4_0_11
Switched to a new branch 'US1234567/CFX-EMM-SYS-Git_2_4_0_11'
jwalla5075k@COTNPL-CWHC372 MINGW64 ~/git-repos/test/CFX-EMM-SYS-Git (US1234567/CFX-EMM-SYS-Git_2_4_0_11)
$ git branch -a
US1234567/CFX-EMM-SYS-Git_2_4_0_11
master
remotes/origin/2016-Goals
remotes/origin/CEMPBAU-672
remotes/origin/CEMPBAU-766
remotes/origin/CEMPBAU-968
remotes/origin/HEAD -> origin/master
remotes/origin/develop
remotes/origin/hotfix-2.1-hf1
remotes/origin/master
remotes/origin/release-2.2
remotes/origin/release-2.5
jwalla5075k@COTNPL-CWHC372 MINGW64 ~/git-repos/test/CFX-EMM-SYS-Git (US1234567/CFX-EMM-SYS-Git_2_4_0_11)
$ |
```



The Git repo work area is composed of three components:

- The `.git` directory that contains all revisions and branches for the repo
- The local workarea directory that is initiated with the last commit of the current branch. Git considers these **Tracked** files.
- Files that have been added, modified, removed, or changed since the local workarea directory was initiated. Git considers these **Untracked** files. This set of deltas is being **Staged** for the next **Commit**. Once they are **committed**, they are considered **Tracked**.

Refer to [2.2 Git Basics - Recording Changes to the Repository](#)¹²

¹² <https://git-scm.com/book/en/v2/Git-Basics-Recording-Changes-to-the-Repository>

```
MINGW64:/c:/Users/awalla5075k/repos/CFX-EMM-SYS-Git

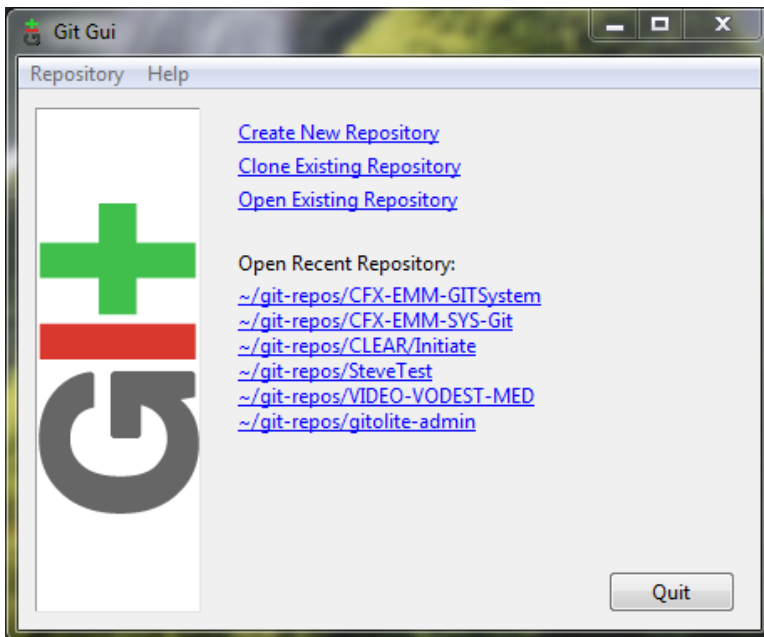
awalla5075k@patap1-8J99L32 MINGW64 ~/repos/CFX-EMM-SYS-Git (master)
$ git branch -a
* master
remotes/origin/HEAD -> origin/master
remotes/origin/NewDocChanges
remotes/origin/develop
remotes/origin/master

awalla5075k@patap1-8J99L32 MINGW64 ~/repos/CFX-EMM-SYS-Git (master)
$ git checkout NewDocChanges
Branch NewDocChanges set up to track remote branch NewDocChanges from origin.
Switched to a new branch 'NewDocChanges'

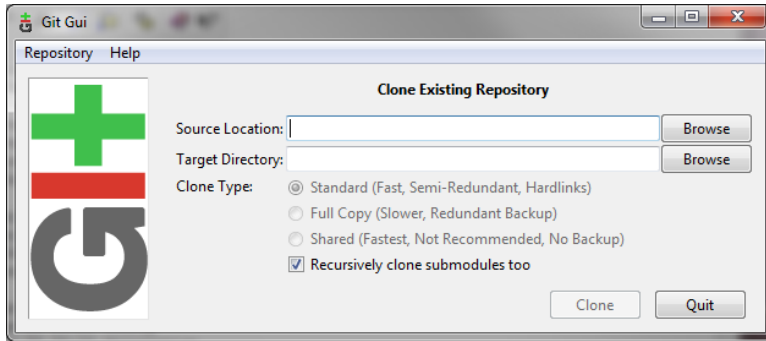
awalla5075k@patap1-8J99L32 MINGW64 ~/repos/CFX-EMM-SYS-Git (NewDocChanges)
$ git checkout -b MyNewDocChanges
Switched to a new branch 'MyNewDocChanges'

awalla5075k@patap1-8J99L32 MINGW64 ~/repos/CFX-EMM-SYS-Git (MyNewDocChanges)
$
```

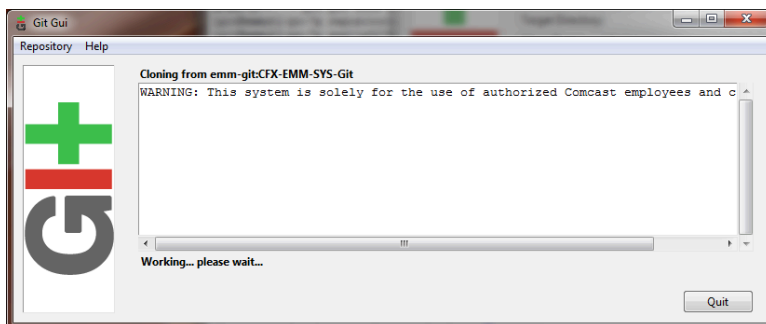
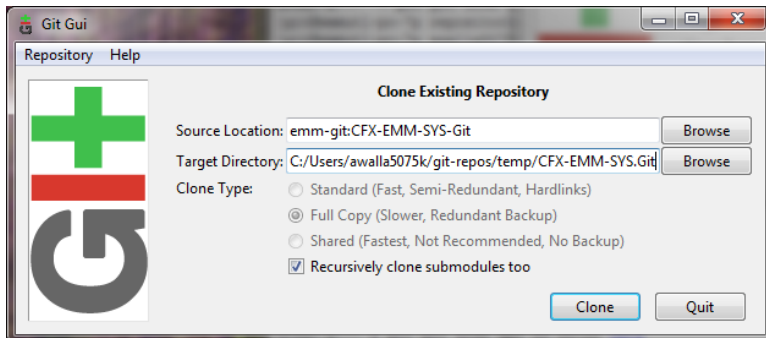
4.3. Using Git GUI:



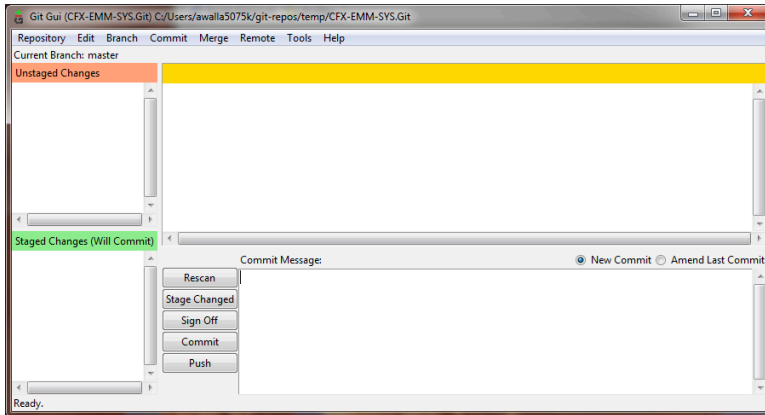
- Select **Clone Existing Repository**



- For **Source Location** enter a Git System repo prepended with ???:
- For **Target Directory**, be sure you are identify a new non-existent directory. It's recommended you name the repo as it is named on the Git System.
- Click on the **Clone** button

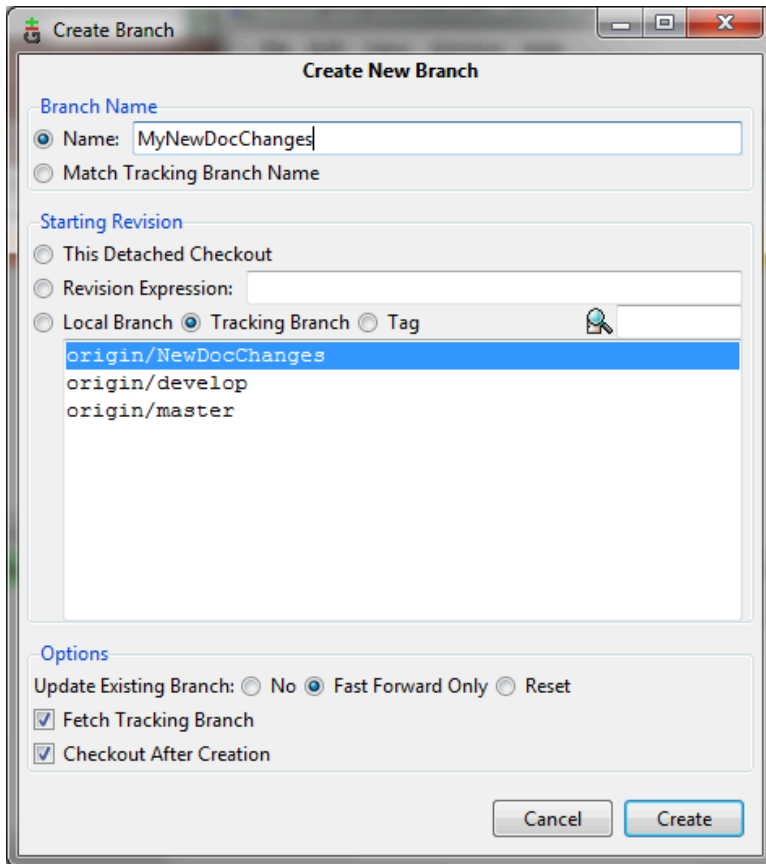


- When the clone is done, the Git GUI will be open in the cloned repo.

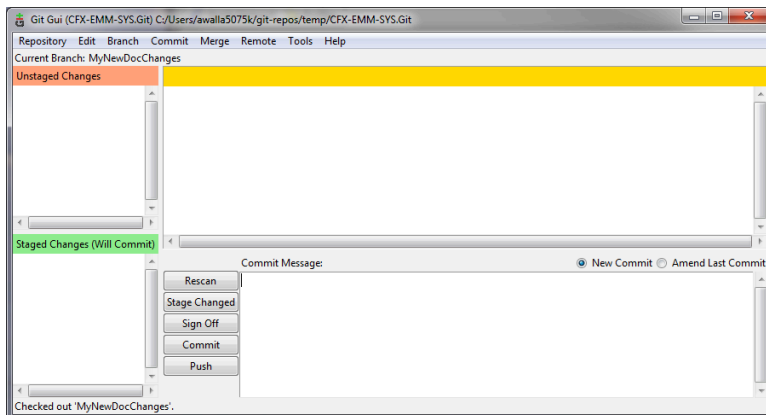


Checkout a New Branch

- To identify and select a branch to work on:
 - Select the Branch pulldown
 - Select Create
 - Select the Name: radio button, Name your branch in the window
 - Select the Tracking Branch radio button
 - Select the branch you wish to checkout
 - Select checkbox **Detach From Local Branch**
 - Click on the **Checkout** button



- Your new branch is now *Checked out*.



At this point you have created a [feature branch] to code on. This can be done with the *Command Line Interface* or with the GUI as indicated above.

If you are working on a JIRA ticket, your feature branch should be named according to the **Development Deployment Branches** section of this document.

After you have made some changes, i.e., creating new files, modifying existing files, and/or removing existing files, and done some testing you need to `git add` your changes so they can be staged for committing to your local feature branch repository. You may want to identify sub-sets of your changes for more testing scenarios. You can limit what you add to the staging environment for your next commit to accomplish this.

Use the `git status` command to keep track of what you have not added and what you have staged for the next commit to your feature branch. Use `git commit` to commit changes to your feature branch.

After each `git commit` on your feature branch, check for conflicts with the [main branch] heads and outstanding release and hotfix branch heads. Repeat the following for each of these branches. Ordering should be master, hotfix, release, develop. The more often you do this, the smaller the amount of potential conflicts:

- Update your local repo for any updates made by others (`git fetch` from the GitHub organization repo and other team profile repos)



Make note of new and updated branches

1. For each new and updated branch execute the following on your feature branch:
 - Merge branch (`git merge --no-ff [branch name]`)
2. Note successful merged changes in the diff reports and address any identified conflicts.



1. **Never code on main branches.**

2. See the *Git Conflict Reporting* section of the [Git Installation Procedure] for a tool that will identify all current conflicts in your local repo.

Now that you have confirmed your new changes do not conflict with any [main branch] heads or outstanding release branches, you are ready to push your changes up to the **GitHub repo** fork in your GitHub profile.



1. The first clone used to establish your workstation repo will name the remote "origin". In this case *git push* will be sufficient since it defaults to the "origin" remote.
2. If you wish to "git push" to an alternate remote, you must specify that remote in the "git push" command.
3. If your feature branch does not yet exist on the EM&M GitHub repo, the git push will respond with a command to establish your branch there. Cut and paste to use it.
4. Only Developer leads can push the **develop** and **master** main branches on the EM&M GitHub organization repo.



This document refers to Git "Porcelain" commands that are build on the Git "Plumbing" commands. Reference [Git Internals - Plumbing and Porcelain](#)¹³ for details.

4.4. GitHub Access Functionality

The [GitHub](#)¹⁴ system is a [bare git repository](#)¹⁵ management system. The protocols established to transfer data between your workstation git repos and the GitHub are:

- Dumb HTTP for read-only access via the GitHub website,
- Secure Shell (SSH) for read-write access.

¹³ <http://git-scm.com/book/en/Git-Internals-Plumbing-and-Porcelain>

¹⁴ <https://github.com/>

¹⁵ <http://www.saintsjd.com/2011/01/what-is-a-bare-git-repository/>

Refer to [The Protocols](#)¹⁶ for details.



It is recommended you configure and use ssh for better security.

In order to execute the procedures in this section, you need have addressed all requirements in the *Prerequisites* section.

5. Is this your First Git push?

Before executing a `git push` for a given GitHub repo, the following questions should help if you are having problems:

1. Did you Review [The Three States](#)¹⁷?
2. Are you familiar with the [Git Reference](#)¹⁸ site?
3. What `git` branch are you attempting to update for the repo in question?
4. What `git` remote are you using for your `git push`?
5. Did you develop on a [feature branch] and merge your changes to your master and develop branches in your local repo?
6. Did you test and verify your `git` merges on your local repo with the `git` repo master and develop branches before using `git push` to github?
7. Did you remove your [feature branch] once it was successfully merged to the develop branch?
8. Are you allowed to `git push` to the remote branch on the GitHub repo?
9. Note that the **develop** and **master** [main branches] on the GitHub repos have restrictions.

The following utilities are available to you when considering the above questions. Be local to your repo on your workstation for the following “git” commands:

Reports on current state of your local repo.

¹⁶ <https://git-scm.com/book/en/v2/Git-on-the-Server-The-Protocols>

¹⁷ <http://git-scm.com/book/en/Getting-Started-Git-Basics#The-Three-States>

¹⁸ <http://gitref.org/>

```
$ git status
```

Lists the current branches in your git repo and any remote repos.

```
$ git branch -a
```

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 GitHub organization.

```
$ git remote -v
```

For your git repo origin remote, shows the git fetch and push URLs, tracked remote branches for your branches and branch push statuses.

```
$ git remote --v show origin
```

6. Git Workflow

In order to understand how branching and merging is arranged into a workflow review [A Successful Git Branching Model by Vincent Driessen](https://nvie.com/posts/a-successful-git-branching-model/)¹⁹. Example git commands related to branching are covered.

Note the Main Branches **develop** and **master** provide for the ability to **vet**²⁰ for conflicts after a `git commit` occurs on any branch at any time.

Since git is a Distributed Version Control System (DVCS) this vetting can occur on the developers git repo without affecting the corresponding bare git repo on the git bare repository system. Each `git push` to the GitHub repo branch must adhere to the following bullets to provide useful accurate vetting to developers:

- All known conflicts at the time of the commit to be pushed have been addressed.
- The commit to be pushed represents a buildable set of code for all environments including upcoming production releases.

¹⁹ [http://nvie.com/posts/a-successful-git-branching-model/](https://nvie.com/posts/a-successful-git-branching-model/)

²⁰ <http://en.wikipedia.org/wiki/Vetting>

- All know tests for the given repo have been executed and modified for current changes in the developer repo on the developer workstation.
- The commit to be pushed represents code that is deployable to all *deployed* to environments.

In order to provide for this vetting, all development is done on feature branches and merged to Main Branches with the merge adhering to the commit rules defined previously.

The GitHub system promotes this with required GitHub forks and pull requests accompanied by DevOps team code reviews.

Refer to [Distributed Git Distributed Workflows](#)²¹ for more information.



1. All merging occurs first on client workstations, not the GitHub system other than controlled code reviews.
2. Developers and developer leads are responsible for managing merge conflicts.
3. For good tips on git commit comments, See [Notes on Contributing Code](#)²² for the github git project.
4. Review [Commit Guidelines](#)²³ to enhance team collaboration.

6.1. Cheat Sheets

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

6.2. Useful Links

- [Commit Often, Perfect Later, Publish Once: Git Best Practices](#)²⁴
- [Undoing Things](#)²⁵
- [Git Software](#)²⁶

²¹ <http://git-scm.com/book/en/v2/Distributed-Git-Distributed-Workflows>

²² <https://github.com/git/git/blob/master/Documentation/SubmittingPatches>

²³ <http://git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project#Commit-Guidelines>

²⁴ <http://sethrobertson.github.io/GitBestPractices/>

²⁵ http://git-scm.com/book/en/v2/Git-Basics-Undoing-Things#_undoing

²⁶ [http://en.wikipedia.org/wiki/Git_\(software\)](http://en.wikipedia.org/wiki/Git_(software))

- [Git Operations²⁷](#)
- [Git reference²⁸](#)
- [The Git Parable²⁹](#)
- [Git from the bottom up³⁰](#)
- [Git for Computer Scientists³¹](#)
- [Understanding Git Conceptually³²](#)
- [git fetch and merge, don't pull³³](#)
- [Git Tutorial³⁴](#)
- [Eclipse Git Tutorial³⁵](#)
- [Git - The simple guide³⁶](#)
- [Why Managed Git Deployment³⁷](#)
- [Cloudways - Using Git for Deployment³⁸](#)

²⁷ http://en.wikipedia.org/wiki/File:Git_operations.svg

²⁸ <http://gitref.org/>

²⁹ <http://tom.preston-werner.com/2009/05/19/the-git-parable.html>

³⁰ <http://ftp.newartisans.com/pub/git.from.bottom.up.pdf>

³¹ <http://eagain.net/articles/git-for-computer-scientists/>

³² <http://www.sbf5.com/~cduan/technical/git/>

³³ <http://longair.net/blog/2009/04/16/git-fetch-and-merge/>

³⁴ <http://www.vogella.com/tutorials/Git/article.html>

³⁵ <http://www.vogella.com/tutorials/EclipseGit/article.html>

³⁶ <http://rogerdudler.github.io/git-guide/>

³⁷ <http://www.cloudways.com/blog/managed-git-deployment/>

³⁸ <https://support.cloudways.com/entries/69615887-Using-Git-for-Deployment->

