[Git Deployment](https://confluence.eu.flextrade.com/display/OMS/Git+Deployment)

[Skip to end of metadata](https://confluence.eu.flextrade.com/display/OMS/Git+Deployment#page-metadata-end)

* Created by [Sagar Gondhali](https://confluence.eu.flextrade.com/display/~sgondhal), last modified by [Abhisang Janrao [X]](https://confluence.eu.flextrade.com/display/~ajanrao" \o ") on [Jan 06, 2021](https://confluence.eu.flextrade.com/pages/diffpagesbyversion.action?pageId=79435817&selectedPageVersions=16&selectedPageVersions=17)

[Go to start of metadata](https://confluence.eu.flextrade.com/display/OMS/Git+Deployment#page-metadata-start)

For Northern Trust we are using git based deployment. It provides following benefits

* Keep track of changes to config files over period of time
* Easily checkout previous version of any/all files
* If accidentally any file is deleted/modified then changes can be reverted back from git repository.

**In git we are tracking following information :**

* Binary files
  + Binary files occupy large storage space and hence are tracked using git-lfs
  + git-lfs speeds up clone and pull operations by fetching only latest copy of files tracked using git-lfs
  + git-lfs also significantly reduces size of local git repo by perodically purging the old lfs files.
* Other files
  + Other files are tracked without git-lfs
    - environment dependent config files : Any config files which will be changed based on server/environment.
      * environment dependent config files are not tracked in git as these config files will change on each environment (DEV/QA/PROD)
      * Environment dependent config files are sym link rather than physical files, these sym links point to environment specific files on each of the setups.
      * e.g. on UAT fc\_40 will point to $FLEXSYS/sys/setup\_flex\_uat/fc\_40 whereas on PROD fc\_40 will point to  $FLEXSYS/sys/setup\_flex\_prod/fc\_40
        + Note : here we don't track $FLEXSYS/sys/setup/fc\_40 in git, instead we track $FLEXSYS/sys/setup\_flex\_uat/fc\_40 and $FLEXSYS/sys/setup\_flex\_prod/fc\_40
      * For more details environment dependent config files and sym link generation check following files on NT setup
        + $FLEXSYS/flex/scripts/generate\_environment\_sym\_links.sh
        + $FLEXSYS/flex/config/servers.conf : Server and folder suffix mapping
        + $FLEXSYS/flex/config/config\_files : List of environment dependent config files
        + $FLEXSYS/.gitignore : As we are not tracking environment dependent files in git we need to add these to .gitignore else these files will show as untracked.
    - Rest of files which don't change based on server/environment are tracked as is.
* Tracking both binary and config files in git allow us to quickly switch server to any given git commit and try out scenarios faced on that versions.

**GitLab URL :**

<http://10.27.0.3/NorthernTrust/NT-OMS-Sys>

**Pushing new changes to git**

Note : Git works best if each person contributing to git has a different work area.

Since all the servers are shared by global NT Implementation teams please make sure following steps are followed. This will make maintenance easier and no one will have to do r&d in order to figure out how to make something work with git.

1. Always use git status command before and after making any change to git
2. Commands to run when pushing new changes
   1. Check status of local git repository :*git status*  
      1. git status output show that there are 3 untracked files
      2. After making some changes to server git status shows following ouptput
   2. Ensure you are working on the head revision: *git pull origin master*
      1. Before you make your changes, ensure the branch has been updated to the current head revision.
      2. This will prevent conflicts when you attempt to commit and push your changes to Git later in the process.
      3. It will also negate the need for you to merge your changes into the head revision.
      4. Note, it is a good idea to perform this on all regional hubs - this will help to identify possible conflicts early before you commit your changes.
   3. *make some changes on server*
   4. Check status of local git repository :*git status*  
      1. In status output git shows that 3 atdl files are locally modified but not yet staged for commit.
   5. discard local changes : *git checkout*
      1. use command *git checkout -- <filename>* in order to discard local changes to any of the file.
      2. This is useful if someone has tried some local binary/config changes which did not work and now these needs to be reverted.
   6. staging changes for commit : *git add*  
      1. use *git add <filename>*command in order to stage required files to staging area
      2. All the git commands support specifying multiple file names
   7. Verify staged files : *git status*  
      1. Verify status of files once staging is done using git add command.
      2. All the staged files will be visible under "Changes to be committed" section
   8. un-stage staged files : *git reset HEAD <filename>*
      1. In order to remove files from staging area use command *git reset HEAD <filename>*
      2. if we unstage all three files from staging area then git status output will be same as 2.c
   9. commit staged files to local git : *git commit*  
      1. Use following command in order to commit changes to git.
         1. git **-c "**[**user.name**](http://user.name/)**=Sagar Gondhali" -c "user.email=Sagar.Gondhali@**[**flextrade.com**](http://flextrade.com/)**"** commit -m "commit message"
      2. Since multiple people share same server it's important to specify author's details using user.name and user.email with each commit.
   10. Verify status and log : *git status and git log*  
       1. atdl files no longer show as cached or locally modified instead we see a different message now "Your branch is ahead of 'origin/master' by 1 commit"
       2. This message means we have one commit on local git repo which does not exist on remote gitlab server.
   11. Check which all commits are not in remote gitlab repository : *git cherry*  
       1. git cherry command can be used to check differences in local git and remote git repository
       2. you can also do it by using git log command and manually matching commit id/messages with remote commits. [<http://10.27.0.3/NorthernTrust/NT-OMS-Sys/commits/master>]
       3. Above output shows that our commit has not made it to remote git repository yet.
   12. Pushing changes to remote repository : *git push*
       1. git is distributed version control system, it maintains multiple copies of git repository. ( One copy on gitlab server and one copy one each of the server where we have done git pull/clone)
       2. All the changes done till now and git commit does not make any changes on gitlab server. All these changes are done on local git repository.
       3. using git push command we need to push these changes to remote repository
       4. git push failure :   
          1. As in following screenshot git push may fail if your local repository does not have all the commits present in remote gitlab repository.
          2. This most likely happens if we don't do git pull before starting adding any changes to git
          3. You may run into same issue if someone else pushed some changes to git between your git pull and git push commands
          4. **workaround :** on server run command *git pull origin master*in order to sync changes from remote git   
             1. On running git pull command, git will open vi editor and ask for a merge message.
             2. Merge prompt is received as git is trying to merge upstream changes  to local commit. Merge prompt won't be received if upstream is changed and there are no local commits in local git.
             3. Type in merge message and save-quit the editor
   13. Sync your changes to all regional hubs: *git pull origin master*
       1. This will ensure all regional hubs are in sync and up to date.
       2. It will prevent other users running into conflicts when they need to make changes.

**Upgrading Package on UAT/Prod:**

* In order to upgrade package on UAT/PROD server download QA package on US UAT server.
  + In case of planned release, QA certified package details are provided by India QA Team
  + For adhoc releases download QA build from primesys to US UAT and execute all 4 scripts
    - Server : [primesys@10.13.37.176](mailto:primesys@10.13.37.176)
    - Password : flexsys123
    - Path : /home/primesys/Releases
    - Scripts :
      * nnosys\_AS7\_DEC\_1710\_QA\_master.sh
      * app\_AS7\_DEC\_1710\_QA\_master.sh
      * api\_AS7\_DEC\_1710\_QA\_master.sh
      * sim\_AS7\_DEC\_1710\_QA\_master.sh
  + webSysconfig and webMOS is always provided by India QA, in case of Adhoc release we won't get webSysconfig and webMOS
* Executing these 4 scripts will upgrade package on US UAT.
* webSysconfig and webMOS packages are provided as tar.gz, extract tar ball on home path
  + After extracting the tar balls check what all files are modified using status command
  + **Note :** We don't use default start and stop scripts provided as part of base package, check what changes are present in start/stop scripts using git diff command. If only difference is logging related then checkout our script from git else merge script changes.
  + **Note :**webSysconfig and webMOS provide env dependent settings as tmp files. We copy entities.json.tmp as is to entities.json. For rest of the files compare .tmp file with one without extension and manually merge data.
* Push the new QA package to git.
  + When pushing package to git make sure you push only files received as part of package and not all the files present on server.
  + Binaries can be pushed as is
  + Config files/scripts need to be merged manually
* Once package is pushed to git login to other UAT servers and do git pull, this will download new package on all the servers
* Login to all the servers and restart all the processes.
  + $HOME/flex/config/monitoring/scripts/kill\_monit.csh
  + $HOME/flex/config/monitoring/scripts/kill\_logstash.csh
  + $HOME/flex/scripts/kill\_simulator.csh
  + $HOME/flex/scripts/kill\_daily.csh SKIP
  + $HOME/flex/scripts/kill\_ioi.csh
  + $HOME/flex/scripts/kill\_weekly.csh SKIP
  + $HOME/webSysConfig/stop\_syscfg.csh
  + $HOME/flex/scripts/start\_weekly.csh SKIP
  + $HOME/flex/scripts/start\_daily.csh SKIP
  + $HOME/webSysConfig/start\_syscfg.csh
  + $HOME/flex/scripts/start\_ioi.csh
  + $HOME/flex/scripts/start\_simulators.csh
  + $HOME/flex/config/monitoring/scripts/run\_flex\_monit.csh
  + $HOME/flex/config/monitoring/scripts/run\_flex\_logstash.csh
* Once all processes are up verify logs and make sure there are no errors
* Verify db Server log and check if there are any exceptions due to schema mismatch

**List of Commands to be used when pushing new changes to git**

1. Add file to git or Modify existing file : git add <file1> <file2>
2. Remove files from git : git rm <file1> <file2>
3. Verify Status : git status
4. commit changes : git **-c "**[**user.name**](http://user.name/)**=<name>" -c "user.email=<email>"** commit -m "<commit message>"
5. push changes to remote git : git push origin master
6. pull changes from remote : git pull origin master

For any further details on git please check Pro Git e-book at  <https://git-scm.com/book/en/v2>

**Pending/Future Items in deployment:**

1. start use of branching for implementation and deployment
   1. Currently we have single (master) branch in git, which means any new development is pushed to master branch.
   2. We may run into a issue when some breaking change is pushed to master which we don't want to pull that to UAT/PROD but want to pull some other latest changes. [This can't be done due to single branching model]
   3. In order to avoid this issue we need to move to branching model of git
      1. develop branch : All development will happen and pushed to develop branch. All dev servers will point to develop branch
      2. master branch : Once any feature is developed and unit tested it will be merged to master branch
      3. UAT servers will point to master branch, any changes on UAT/master branch should be stable as these are already tested on develop branch.
      4. In case if any bugs are observed on UAT and needs to be fixed then these should be done in 2 places
         1. fix the bug in master branch (UAT) and push it to git
         2. Apply patch to develop branch and push to git
2. Reference for git flow workflow : <https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow>