CATS Gitflow Branch Management Process Guideline

Background

Git repository usage for CATS is based on topic/feature branch strategy. It dictates topic branch development approach with a handful of long-lived mainline branches. These branches are:

master: branch where production build is created from. This is read-only branch and should never be pushed to directly.

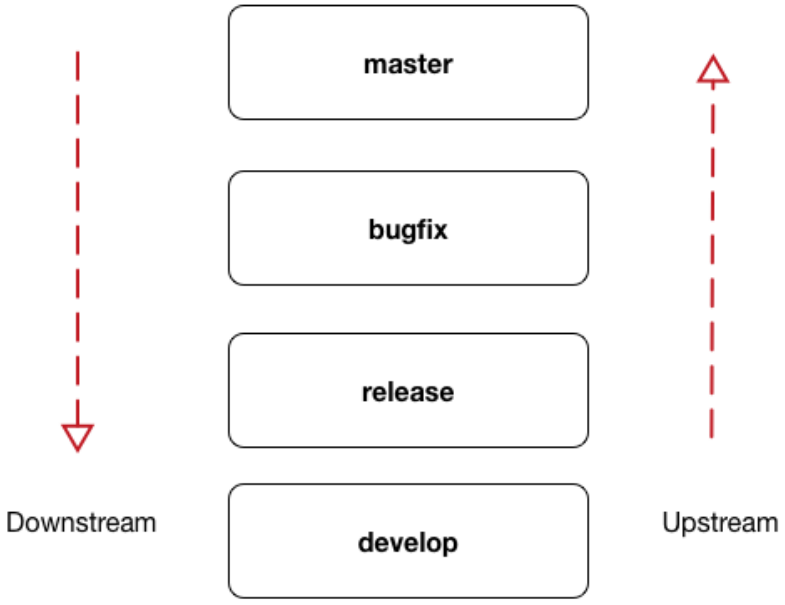
bugfix: contains near identical codebase to that of master but also used for implementing production bug fixes and urgent feature implementations.

release: code base for upcoming new features being tested and primed for CATS vNext

develop: where bleeding edge feature feature implementation takes place.

In order to have branch changes in sync and keep conflicts to a minimum it is a good practice to do frequent merging between mainline branches. Merges could be either upstream or downstream depending on what branch you consider as the baseline. It is mandatory to define a convention for communicating direction of merges when doing upstream or downstream merge activities.

Branch merging naming conventions



Setting up your environment for branch management

1. Clone CATS repository from Github into your preferred directory.

   ```
   git clone https://github.com/ndrmc/cats.git
   ```

2. Make sure that your git information is configured

   git config --global user.name "Your Name"

   git config --global user.email email@address.com

3. Install and configure diff and merge UI tool. There are a number of tools out there but the two popular ones are WinMerge (winmerge.com) and Meld (meldmerge.org). Install and configure your preferred merge tool on your machine.

Configuring diff tool

In order to use diff/merge tool with git, you first need to

1. Once the installation is complete open your git configuration file found at your home directory called (.gitconfig) and add the following lines:

```
[mergetool]
    prompt = false
    keepBackup = false
    keepTemporaries = false

[merge]
    tool = winmerge


[mergetool "winmerge"]
    name = WinMerge
    trustExitCode = true
    cmd = "/c/Program\\ Files\\ \\(x86\\)/WinMerge/WinMergeU.exe" -u
-e -dl \"Local\" -dr \"Remote\" $LOCAL $REMOTE $MERGED


[diff]
    tool = winmerge


[difftool "winmerge"]
    name = WinMerge
    trustExitCode = true
```

```
        cmd = "/c/Program\\ Files\\ \\(x86\\)/WinMerge/WinMergeU.exe" -u
    -e $LOCAL $REMOTE
```

Note that the above configuration is for WinMerge and you need to change appropriate paths and file names if you're using another tool.

Three way merging

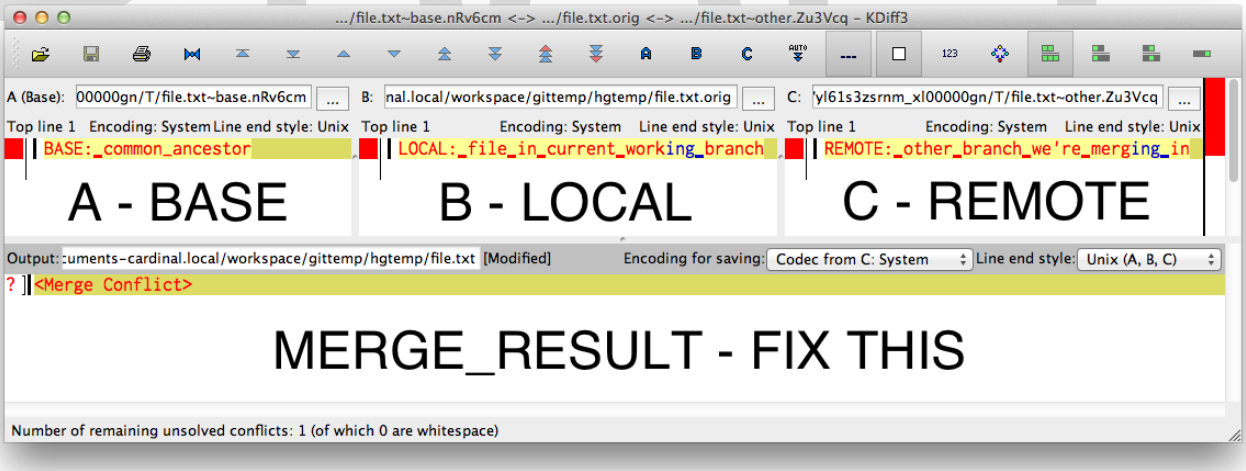
Git supports three-way merging out of the box. Unlike diff, three-way-merge has the concept of Local, Remote, Base and Merge result. Whenever you do a merge to your current branch by issuing the command git merge REMOTE_BRANCH --no-ff git tries to identify each sides using the following naming:

LOCAL - the current branch you're working on and the branch changes are being merged to

REMOTE - the head of remote repo/branch that is going to be merged

BASE - common ancestor to both LOCAL and REMOTE

MERGED - the file that will be written as a result of the merge.



Downstream merging of branches

Downstream merging is the process of moving changes from higher branches to lower ones with the purpose of reducing merge conflicts by conducting merging on regular basis (preferably daily). Everyday mainline branches are merged downstream to avoid a situation where by branches divert from each other.

Downstream merge from bugfix to release

This activity should be conducted on regular interval or at least whenever a bug is resolved on **bugfix** branch. The reason is since the next batch of new features is being implemented on **release** branch we want to ensure that this fix makes it to the next version and not only to the current production one.

1. Checkout **bugfix** branch and pull recent changes from Github

   ```
   git checkout bugfix

   git pull origin bugfix
   ```

2. Checkout release branch and pull recent changes

   ```
   git checkout release

   git pull origin release
   ```

3. While on release branch, merge bugfix

   ```
   git merge bugfix --no-ff
   ```

4. If there are no merge conflicts git will ask you to provide a commit message using your configured editor (defaults to vim)
5. If there are merge conflicts git will stop the process and allows you to resolve conflicts

6. To resolve merge conflicts (assuming that you have configured merge tool on your machine) type

```
git mergetool
```

7. Repeat this step until all conflicting files are resolved
8. When all conflicts are resolved add modified files to git stage and commit your changes

```
git add .

git commit -m “Merge bugfix -> release”

git push origin release
```


Downstream merge from release to develop

Once you completed downstream merging of bugfix to release continue to merge changes to develop branch. This is required because we also want to include all bug fixes and changes made in bugfix and release branch to be part of develop branch too.

1. Checkout release branch and pull recent changes from Github

```
git checkout release

git pull origin release
```

2. Checkout develop branch and pull recent changes

```
git checkout develop

git pull origin develop
```

3. While on develop branch, merge release to develop

```
git merge release --no-ff
```

4. If there are no merge conflicts git will ask you to provide a commit message using your configured editor (defaults to vim)
5. If there are merge conflicts git will stop the process and allows you to resolve conflicts
6. To resolve merge conflicts (assuming that you have configured merge tool on your machine) type

```
git mergetool
```

7. Repeat this step until all conflicting files are resolved
8. When all conflicts are resolved add modified files to git stage and commit your changes

```
git add .

git commit -m "Merge release -> develop"

git push origin develop
```


Upstream (Upward) branch management

Every time new versions are about to be released; an upstream branch merging is triggered. This also flags the start of testing effort on test environments for the CATS vNext as bits which comprise the next version are going to be deployed to test environments. We follow the steps below to conduct upstream branch merging

Creating new release branch

Precondition:

- *CATS Project Manager and Technical Team Development manager have decided (in consultation with PO) the list of features that will make up a specific version.*
- *Topics branches which contain features included in the version to be created have been successfully merged to develop branch.*

1. Merge release branch to develop. Delete existing release branch from remote repo (Github).

```
git checkout release

git pull origin release

git checkout develop

git pull origin develop

git merge release --no-ff

git push origin :release  or

git push origin --delete release
```

2. Create new **release** branch from develop. While on develop branch issue the following command

```
git checkout -b release
```

3. Create tag for the newly created release branch corresponding to milestone version number.

```
git tag x.xx.xx -m “Creating tag for new release branch”
```

4. Push both release branch branch and created tag

```
git push origin release

git push --tags
```

5. Push builds to test environment.
6. Inform team about branch and tag changes.

Doing new production build & moving release to production

Precondition:

- *Testing is completed on release branch (test environment)*
- *Changes made on bugfix branch are merged to release*
- *Team is informed about upcoming production build*

1. Merge all changes that are made in bugfix branch to release.

```
git checkout bugfix

git pull origin bugfix

git checkout release

git pull origin release

git merge bugfix --no-ff
```

2. Resolve any outstanding merge conflicts (if any) [NOTE: Any need to do builds?]
3. Merge **release** branch to **master** in order to stage changes to production environment

```
git checkout master

git pull origin master

git merge release
```

4. Create tag for the new production build version corresponding to milestone number by bumping appropriate version numbers.

```
git tag x.xx.xx -m “Creating tag for new production build”
```

5. Push both release and master branches to remote Github repo.

```
git push origin release

git push origin master
```

6. Push both branches and tags

```
git push origin release

git push origin master

git push --tags
```

7. Trigger production build from tag created in step 4 above. (Refer to build process for more detail)
7. Create release note in the wiki which includes corresponding bugs, support requests and features implemented in the release. Also send out email to members of task force and user mailing list which include link to the wiki.