**[Start the Git server](https://stark.itblab.npfit.nhs.uk/display/DOH/Start+the+Git+server)**

[Skip to end of metadata](https://stark.itblab.npfit.nhs.uk/display/DOH/Start+the+Git+server#page-metadata-end)

* Added by [Ian Fawcus](https://stark.itblab.npfit.nhs.uk/display/%7Eianf), last edited by [Ian Fawcus](https://stark.itblab.npfit.nhs.uk/display/%7Eianf) on May 15, 2015

[Go to start of metadata](https://stark.itblab.npfit.nhs.uk/display/DOH/Start+the+Git+server#page-metadata-start)

The Git server is hosted on the Frey server: 10.210.164.102

SSH onto the server as iam\_user - the password is Careid1

In the /home/Iam\_user directory run tthis command to start the server:

|  |
| --- |
| java -server -Xmx1024M -jar gitblit.jar & |

# [GIT branching model](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+branching+model)

[Skip to end of metadata](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+branching+model#page-metadata-end)

* Added by [Satyaraj Thakor](https://stark.itblab.npfit.nhs.uk/display/%7Esat), last edited by [Satyaraj Thakor](https://stark.itblab.npfit.nhs.uk/display/%7Esat) on Sep 19, 2013  ([view change](https://stark.itblab.npfit.nhs.uk/pages/diffpages.action?pageId=4162030&originalId=4164163))

[Go to start of metadata](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+branching+model#page-metadata-start)

All you need to know about GIT Branching Model for CareID development. The following strategy is based upon Git Flow model designed and detailed by Vincent Driessen in the following blog post <http://nvie.com/posts/a-successful-git-branching-model/>

### All Branches

* master
* develop
* feature
* release
* hotfix

### The Main Branches

The following are the 2 main branches with infinite timeline...

* master
* develop

The **master** is the main branch, of which the head source code always reflects a production ready state after every release. As for the developer is concerned, this branch is almost untouchable. The **feature** branches controlled by the developer would merge back into the **develop** branch, as explained below.

The **develop** is a branch of **master** and runs in parallel to the **master** branch that reflects the latest delivered development changes for the current release. This branch is used specifically for development and CI processes. Developers would create **feature** branches from **develop** and after completion of the feature, will merge the code back in **develop**. CI builds will execute off the **develop** branch, hence the developer should make sure that the merged code builds successfully.

The **release** branches are used to prepare releases as part of a formal process. As a developer you may not need to be concerned with this process or these branches. When preparing a release, a **release** branch is created from the **develop** branch. Meta-data for the release and any last minute bug fixes are committed onto the **release** branch. When the release is finalised the **release** branch is merged into **master** and back into **develop** - making **master** reflect the new state of production ready code and ensuring that any code changes that happened on the **release** branch are captured back on **develop**.

The **hotfix** branch will be used for the fixing issues in the released code that resides in the **master** branch

**Tags** are used on the **master** branch to flag the commit that represents each release or hotfix.

Tags are used on the **develop** branch to flag each successful CI build with the build number, to support looking at the specific code base that a bug was found in (since bugs are raised against the build number they were found in).

### Setting up

To setup GIT and GIT Flow on your workstation see the link below.

* [Setting up GIT and GIT Flow](https://stark.itblab.npfit.nhs.uk/display/IAM/Setting+up+GIT+and+GIT+Flow)

### Create and manage feature branch

The developer will create **one GIT feature branch for each JIRA Issue** assigned. The lifetime of this feature branch is as long as the developer is working on the JIRA issue. When the development is finished the feature branch will be merged back into develop branch and remove the feature branch from the repository.

The benefits of creating is a feature are;

* to isolate from other developers work and the develop branch which is always in a buildable, clean state
* the upstream repository always has the code you are working on, which acts as a backup and can be picked up by others working on the same feature

The following lists down the steps to be carried out to implement the branching strategy

* [Create Feature Branch](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference#GITCommandsReference-CreateFeatureBranch)
* [Manage Feature Branch during development](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference#GITCommandsReference-ManageFeatureBranch)
* [Committing Feature Branch](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference#GITCommandsReference-CommitFeatureBranch)
* [Keep Feature branch up-to-date with Develop branch](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference#GITCommandsReference-UpdateFeatureWithDevelop)

### When feature finishes

You should keep your feature branch up to date with develop as much as possible. The more frequently you merge from develop the easier the merge will be and the fewer problems you will have.  
Don't merge from Master or Release branches, source code should always reach your feature branch through a merge from the develop branch. Sometimes you may need to merge from one feature branch to another feature branch, this is OK, but you shouldn't do it as a matter of routine. The normal process is to:

* [Ensure all changes are committed to your feature branch](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference#GITCommandsReference-CommitFeatureBranch)
* [Merge from develop to bring your feature branch up to date](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference#GITCommandsReference-UpdateFeatureWithDevelop)
* Execute a Feature branch build on Jenkins
* Resolve any issues
* [Merge from develop into your feature branch to take other people's recent commits](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference#GITCommandsReference-MergeFeatureIntoDevelop)
* [Finish your feature branch merging back into develop](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference#GITCommandsReference-MergeFeatureIntoDevelop)
* This will automatically kick off a CI build on Jenkins, watch for the build outcome

# [Setting up GIT and GIT Flow](https://stark.itblab.npfit.nhs.uk/display/IAM/Setting+up+GIT+and+GIT+Flow)

[Skip to end of metadata](https://stark.itblab.npfit.nhs.uk/display/IAM/Setting+up+GIT+and+GIT+Flow#page-metadata-end)

* [Attachments:7](https://stark.itblab.npfit.nhs.uk/pages/viewpageattachments.action?pageId=4163530&metadataLink=true)
* Added by [Satyaraj Thakor](https://stark.itblab.npfit.nhs.uk/display/%7Esat), last edited by [Ian Fawcus](https://stark.itblab.npfit.nhs.uk/display/%7Eianf) on May 15, 2015  ([view change](https://stark.itblab.npfit.nhs.uk/pages/diffpages.action?pageId=4163530&originalId=16908584))

[Go to start of metadata](https://stark.itblab.npfit.nhs.uk/display/IAM/Setting+up+GIT+and+GIT+Flow#page-metadata-start)

### Setting up GIT and GIT-Flow (on Windows 7 workstation)

### ****Important: Please ensure that all steps are followed accurately and fully.****

### ****Note: The process is involved, so if you have any issues please ask**** [Sat](mailto:satyaraj.thakor@hscic.gov.uk) ****or**** [Brett](mailto:brett.jackson@hscic.gov.uk) ****for help.****

GIT Flow is an open source project that facilitates the process explained above with "short hand" commands. GIT flow is to be installed on every client machine that will be used to push to the upstream GIT server.

GIT Flow does not have an installer for Windows, so you will need to follow the steps described in this document.

**Pre-requisites:**

* A Windows 7 (x64) or Windows 8 (x64) device.  32-bit Windows is not supported.
* Privileged or open Internet access, allowing download of the components linked on this page.
* Local administrative privileges.
* Access to this page from the system on to which you are installing.  You may need add a hosts file entry:
  + On Windows, the hosts file is C:\Windows\System32\drivers\etc\hosts

10.210.164.101 stark.itblab.npfit.nhs.uk

**Process**

1. Download and install [Git For Windows](http://git-scm.com/download/win)
   1. Install using all the default options - it will install GIT to "C:\Program Files (x86)"
2. Download and install [p4merge](https://stark.itblab.npfit.nhs.uk/download/attachments/4163530/p4vinst64.exe?version=1&modificationDate=1382611320000&api=v2)
   1. Install using all the default options
   2. Installing TortoiseGIT is optional and just another tool with a different look and feel for diff and merging.
   3. Download and Install [TortoiseGIT](https://stark.itblab.npfit.nhs.uk/download/attachments/4163530/TortoiseGit-1.8.6.0-64bit.msi?version=1&modificationDate=1383583641000&api=v2)
3. Restart windows explorer so it knows about git being on your path
4. Using notepad (or similar), edit the "C:\Program Files (x86)\Git\etc\gitconfig" file and paste the following alias and mergetool stanzas to the end.  Save the changes.

|  |
| --- |
| [alias]      new-feature-web = "!sh -c \"git branch -r | grep -q feature/$1 && echo \\\"Remote branch origin/feature/$1 already exists, you must use a different name\\\" || (git flow feature start $1; git flow feature publish $1; C:/jenkinsJobCreate/create-feature-web-job.sh $1)\""      new-feature-auth = "!sh -c \"git branch -r | grep -q feature/$1 && echo \\\"Remote branch origin/feature/$1 already exists, you must use a different name\\\" || (git flow feature start $1; git flow feature publish $1; C:/jenkinsJobCreate/create-feature-auth-job.sh $1)\""      new-feature-cms = "!sh -c \"git branch -r | grep -q feature/$1 && echo \\\"Remote branch origin/feature/$1 already exists, you must use a different name\\\" || (git flow feature start $1; git flow feature publish $1; C:/jenkinsJobCreate/create-feature-cms-job.sh $1)\""    [merge]      tool = p4merge  [mergetool "p4merge"]      path = "C:/Program Files/Perforce/p4merge.exe"    [mergetool "tortoisemerge"]      cmd = \""c:/Program Files/TortoiseGIT/bin/TortoiseGitMerge.exe"\" -base:"$BASE" -theirs:"$REMOTE" -mine:"$LOCAL" -merged:"$MERGED"    [diff]      tool = tortoisediff  [difftool "tortoisediff"]      cmd = \""c:/Program Files/TortoiseGIT/bin/TortoiseGitMerge.exe"\" -mine:"$REMOTE" -base:"$LOCAL" |

2. Create a new folder C:\jenkinsJobCreate
3. Modify the NTFS permissions to give the **Everyone** group Read & Modify
   1. Open a new **Windows Explorer** window, and navigate to C:\
   2. Expand **Local Disk (C:)**
   3. Right click **jenkinsJobCreate** and click **Properties**
   4. Select the **Security** tab
   5. Click **Edit...**
   6. Select **Everyone** and tick **Modify** (this will ensure other applicable permissions are also ticked)
   7. Click **OK**
   8. Click **OK** again
4. Download the following files and save them to the C:\jenkinsJobCreate folder
   1. [create-feature-web-job.sh](https://stark.itblab.npfit.nhs.uk/download/attachments/4163530/create-feature-web-job.sh?version=1&modificationDate=1379514761000&api=v2)
   2. [create-feature-auth-job.sh](https://stark.itblab.npfit.nhs.uk/download/attachments/4163530/create-feature-auth-job.sh?version=1&modificationDate=1379514772000&api=v2)
   3. [create-feature-cms-job.sh](https://stark.itblab.npfit.nhs.uk/download/attachments/4163530/create-feature-cms-job.sh?version=1&modificationDate=1379514783000&api=v2)
   4. [jenkins-feature-config.xml](https://stark.itblab.npfit.nhs.uk/download/attachments/4163530/jenkins-feature-config.xml?version=1&modificationDate=1379511340000&api=v2)
5. Edit each of the 3 .sh files mentioned above and change the localhost port for the jenkins server
   1. This should be 8080 if you are going to be using Lannister for your repo
6. Download [getopt.exe](https://stark.itblab.npfit.nhs.uk/download/attachments/4163530/getopt.exe?version=1&modificationDate=1379599049000&api=v2) and copy it to C:\Program Files (x86)\Git\bin
7. Download and install msysgit to a location of your choice
   1. You can now find it at the bottom of the page at <http://msysgit.github.io/>
   2. Try <https://github.com/msysgit/msysgit/releases/download/Git-1.9.4-preview20140929/msysGit-netinstall-1.9.4-preview20140929.exe>
8. To clone git-flow from git
   1. Open a Windows Command Prompt and issue the following commands

**command window as your user on Windows**

|  |
| --- |
| cd /d c:\  mkdir c:\gitflow  cd gitflow  git clone https://github.com/petervanderdoes/gitflow.git  cd gitflow  contrib\msysgit-install.cmd "C:\Program Files (x86)\Git" |

1. Download and install [GNUWin32](http://sourceforge.net/projects/getgnuwin32/files/):
   1. Execute the exe and install at an appropriate location, i.e.
      1. **C:\Program Files (x86)\GnuWin32\**
   2. From windows command prompt navigate to the prompt execute download.bat (this will take a while)
   3. Execute install.bat
2. Set the Windows environment path variable to include the path to new tools
   1. Open Windows Explorer
   2. Right click **Computer** and select **Properties**
   3. Click **Advanced systemsettings**
   4. Select the **Advanced** tab and click the **Environment Variables...** button
   5. Under **System Variables**, find the **Path** variable and click **Edit...**
   6. Append "**;C:\Program Files (x86)\Git\bin;C:\Program Files (x86)\GnuWin32\bin;"** (ensure you include the leading semi-colon) to the end of the **Variable** value, and be careful not to corrupt the existing value.
      1. This must match the installs you did above!
   7. Click **Ok**, click **Ok** again, then finally click **Ok** once more
   8. Restart your windows machine
3. To get started with GIT flow on your machine
   1. Open a GitBash command prompt
   2. Use the following commands to clone the **master** branch or make use of eGIT eclipse plugin to clone the CareID\_Development repository
   3. tunnelPortForGit = 1085 for lannister
   4. Like this:

**gitbash as your user on Windows**

|  |
| --- |
| git config --global user.name 'Your\_GIT\_User\_Name\_no\_spaces'  git config --global user.email you@somedomain.com  git config --global push.default current    git clone ssh://iam\_user@10.210.164.102/home/iam\_user/gitblit/data/git/CareID\_Development.git /c/git/CareID\_Development  cd <project folder>  git branch  #this should display "master" at this stage    git checkout develop  #this is to initialise the git flow model on your machine>  #Please select all the default options and continue setup  git flow init    # git branch  #this should now display "develop" as your current active branch and a master branch    git ls-remote  #this command will show the branches on remote head    git clone ssh://iam\_user@10.210.164.102/home/iam\_user/gitblit/data/git/CareID\_Auth.git /c/git/CareID\_Auth  cd <project folder>  git branch  #this should display "master" at this stage    git checkout develop  #this is to initialise the git flow model on your machine>  #Please select all the default options and continue setup  git flow init    # git branch  #this should now display "develop" as your current active branch and a master branch    git ls-remote  #this command will show the branches on remote head |

[**GIT Commands Reference**](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference)

[Skip to end of metadata](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference#page-metadata-end)

* [Attachments:1](https://stark.itblab.npfit.nhs.uk/pages/viewpageattachments.action?pageId=4163549&metadataLink=true)
* Added by [Satyaraj Thakor](https://stark.itblab.npfit.nhs.uk/display/%7Esat), last edited by [Satyaraj Thakor](https://stark.itblab.npfit.nhs.uk/display/%7Esat) on Dec 18, 2013  ([view change](https://stark.itblab.npfit.nhs.uk/pages/diffpages.action?pageId=4163549&originalId=7471664))

[Go to start of metadata](https://stark.itblab.npfit.nhs.uk/display/IAM/GIT+Commands+Reference#page-metadata-start)

This is list of GIT commands.

**Quick Reference Cards**

[Click here to download quick reference cribsheet](https://stark.itblab.npfit.nhs.uk/download/attachments/4163549/GIT-Cribsheets.pdf?version=2&modificationDate=1390581353000&api=v2).

**1. Create Feature Branch**

Though we are using the GIT Flow to create feature branch, there are 3 GIT Aliases defined;

* new-feature-web
* new-feature-auth
* new-feature-cms

These alieases are used to create feature branch that makes use of GIT Flow commands and also performs;

* checks for the feature branch name uniqueness
* publishes feature branch to upstream repository
* creates a Jenkins job for the feature

Use GIT Bash to issue the following commands

|  |
| --- |
| Ensure your current branch is clean - no unstaged files, no added/removed/modified files    Checkout develop  # git checkout develop    Bring your develop branch up to date  # git pull    Issue this command to create a new feature branch - do not include feature/ on the start  # git new-feature-web dev1-US0021-createUser |

**The feature branch naming** convention is:

<Developer Initials>-<JIRA Issue Num>-<Brief Feature Name>

**2. Manage Feature Branch during development**

For GIT command line reference see <http://git-scm.com/book/en/Git-Basics-Recording-Changes-to-the-Repository>

**2.1 Commit Feature Branch**

It is a good practice to keep committing the changes to your feature branch and push changes to the upstream server (using "push") at least ones at the end of the day. After having staged all modified files for commit, issue the following command;

|  |
| --- |
| # git commit -m "Short description of the changes"  # git push |

**2.2 Update Feature branch with Develop branch**

You should periodically bring your Feature branch up-to-date with the latest code on the Develop branch. This should be done at least once a day, and ideally after every successful CI build. The more frequently you merge changes from Develop, the easier those merges will be.

In Git there are several ways to bring changes from one branch into another. Our process is to do a merge; **do not use of rebase**.

|  |
| --- |
| Ensure you have committed everything on your Feature branch    # git push  # git pull origin develop    Deal with any merge conflicts, then push to remote repository  # git commit -m "Short description of the merges"  # git push |

**2.3 Merging Feature Branch into Develop Branch**

**\*\*\* This is the ONLY way you should be changing the Develop branch. Never edit files directly on the Develop branch, and never rebase the Develop branch \*\*\***

|  |
| --- |
| # git push  # git checkout develop  # git pull  # git flow feature finish dev1-US0021-createUser  # git push |

The feature finish command does the following:

1. Merge feature branch into develop
2. Removes the local feature branch
3. Switches local repository to develop branch

If you have conflicts when Git-flow merges your Feature branch into Develop (you shouldn't if you follow the process correctly), then you will have to resolve these conflicts on your Feature branch, commit the changes, and do the "feature finish" command again.  
Note this will delete the feature branch locally, but at this stage it still exists in the remote repository. To delete the feature branch in the remote repository too:

|  |
| --- |
| # git delete-remote feature/dev1-US0021-createUser |

2.4 Resolving Merge Conflicts

After a merge you may get unresolved merge conflicts which Git could not automatically resolve. This is usually because two people have edited the same part of a file.  
To see what unresolved conflicts you have (they will be listed under the "unmerged" section):

|  |
| --- |
| # git status |

Files those are in conflict have standard conflict resolution markers added to them. You can manually edit the file to resolve this, but it is better to use the tools provided

|  |
| --- |
| # git mergetool |

This will open the tool p4merge to allow you to resolve the merge conflict, by selecting between blocks. You may also edit the resulting merged file, by manually editing the contents of the bottom panel should you need to. If there are muliple unmerged files, p4merge will cycle through them in turn - as you close the p4merge window for the first conflict it will open again for the next. After saving in p4merge and closing it, Git will ask you if the merge was successful, if it was it will mark the conflict as resolved by automatically staging the file for you.

If you have manually resolving a merge conflict (not using git mergetool) you must also manually mark it as resolved by staging it:

|  |
| --- |
| # git add <file\_that\_i\_manually\_resolved> |

After you have resolved the conflicts, run git status again to double check there are no outstanding conflicts.

You must then commit to finalise the merge:

|  |
| --- |
| # git commit |

**3. Release Process**

The following describes the steps to perform GIT release

|  |
| --- |
| # git checkout develop  # git flow release start <name\_of\_the\_release>  # git flow publish <name\_of\_the\_release>  # git pull  <perform last minute changes and fixes and release docs>  # git commit -m "comments"  # git push  <ideally, the following step should not be necessary, because you would not expect anybody to be updating develop branch at this moment>  # git pull origin develop  # git push  # git flow release finish <name\_of\_the\_release>  # git push  # git push --tags  # git checkout master  # git pull  # git push |

**4. Issue Resolution**

| **Error** | **Description** | **Resolution** |
| --- | --- | --- |
| git branches have diverged | Not able to finish feature.  Switched to feature branch and performed git pull resulted in the error (not linked to remote branch) | git branch --set-upstream-to=origin/<remote-feature> <local-feature> |
|  |  |  |

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

[**JIRA / Git Flow Cheatsheet**](https://stark.itblab.npfit.nhs.uk/pages/viewpage.action?pageId=15008794)

[Skip to end of metadata](https://stark.itblab.npfit.nhs.uk/pages/viewpage.action?pageId=15008794#page-metadata-end)

* Added by [Matt Dean](https://stark.itblab.npfit.nhs.uk/display/%7Emattd), last edited by [Matt Dean](https://stark.itblab.npfit.nhs.uk/display/%7Emattd) on May 27, 2015  ([view change](https://stark.itblab.npfit.nhs.uk/pages/diffpages.action?pageId=15008794&originalId=17858951))

[Go to start of metadata](https://stark.itblab.npfit.nhs.uk/pages/viewpage.action?pageId=15008794#page-metadata-start)

**Most importantly, read this:** [**http://nvie.com/posts/a-successful-git-branching-model/**](http://nvie.com/posts/a-successful-git-branching-model/)

*The following is just a TL;DR. It's all covered in the above. If you've not read it you don't really know what you're doing.*

**Start work on a new JIRA Story**

If you need to work on something that's not in JIRA, add it to JIRA first.

Create a new feature branch, **branched off 'develop'**

eg for JIRA Story CAR-666 '*Fix XYZ because blah blah*'

|  |
| --- |
| git checkout develop; git pull  git checkout -b car-666\_fix\_xyz develop  git push -u origin car-666\_fix\_xyz |

**Move a JIRA Story into Test**

Ensure your Story's feature branch is up to date with the current Develop

|  |
| --- |
| git checkout develop ; git pull  git checkout YOUR\_FEATURE\_BRANCH  git merge develop |

Then resolve any conflicts and ensure your functionality still works, \*then\* hand over to Test.

**Finish a JIRA Story**

If you finish a story that's not scheduled to be in the next release in JIRA, don't merge to develop.

Once the feature branch is Done (i.e. has been dev reviewed and tested) **and** it's marked in JIRA as in the next release, then **merge it back into 'develop'** and push it

|  |
| --- |
| git checkout develop ; git pull  git merge --no-ff YOUR\_FEATURE\_BRANCH  git push origin develop |

**Cut a Release**

Create a release branch from develop

**Create a release branch for release 19**

|  |
| --- |
| git checkout develop; git pull  git checkout -b release-19.0 develop  git push -u origin release-19.0 |

**Update Master after a Successful Prod Release**

**Merge Release-VERSION back into Master and Develop**

|  |
| --- |
| # Merge release-VERSION to master  git checkout master; git pull --tags  git merge --no-ff release-VERSION    # Tag master with the release number  git tag -a VERSION    # Push master  git push origin master --tags    # Merge master into develop and push  git checkout develop; git pull  git merge --no-ff master  git push origin develop |

**Make a Hotfix Release**

A Hotfix is an emergency change that needs to be applied immediately against a Production environment. All other changes (even if you really *want* them) should follow the normal release process.

Whether you need to Monkey Patch a production system or not is not something covered here. It will depend on the nature of the incident. Irrespective, follow the following process for traceability etc.

**Hotfix Master (at release 19)**

|  |
| --- |
| git checkout master; git pull  git checkout -b hotfix-19.0.1 master    # write and test the fix    git commit -m 'Your commit message'  git push -u origin hotfix-19.0.1    # apply branch hotfix-19.0.1 to the production system    # merge the fix into master and push  git checkout master; git pull  git merge --no-ff hotfix-19.0.1    # Tag master with the release number  git tag -a 19.0.1    git push origin master --tags |

Email the whole team to tell them that you've done this. Develop and any in-progress release branches will need updating to include this hotfix or else the change will be regressed by the next normal release















