Git GitHub and Git Bash survival guide

Table of Contents

1 Preventing problems

There are some things you should do before you start pushing or pulling projects

1.1 You will need to allow Gitbash to use long filenames

Run the following command in Command Terminal (cmd.exe)
git config --system core.longpaths true

1.2 If you get a detached Header warning when checking out a branch/stream

Some branches/streams when checking out without a parameter give you a "Detached HEAD".
Checkout a branch using the **-b parameter** so that you don't get a "Detached HEAD":

git checkout **-b SANPAY-29275** remotes/origin/feature/**SANPAY-29275**

git checkout **-t** remotes/origin/feature/**SANPAY-29275** // this also worked

1.3 Ignoring IntelIJ .iml and .idea files in GIT when using IntelliJ

1.3.1 Adding IntelliJ Artefacts to .gitignore Files

add to the .gitignore files

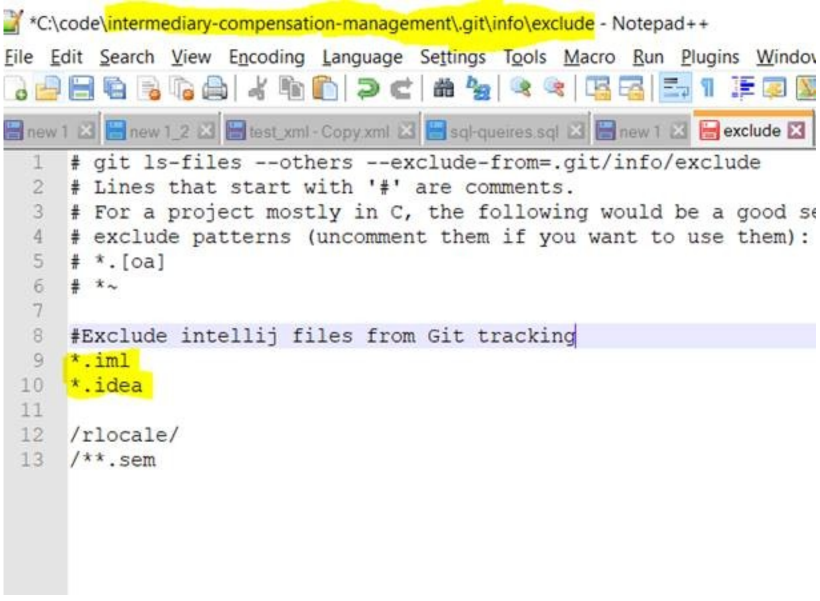
.idea/

**/*.iml

1.3.2 Editing the .git/info/exclude Files and adding IntelliJ Artefacts to ignore

In the local folder …./.git/info edit the exclude file and add these entries:

*.iml
*.idea



*C:\code\intermediary-compensation-management\.git\info\exclude - Notepad++

File Edit Search View Encoding Language Settings Tools Macro Run Plugins Windo
new 1
new 1_2
test_xml - Copy.xml
sql-queires.sql
new 1
exclude

```
# git ls-files --others --exclude-from=.git/info/exclude
# Lines that start with '#' are comments.
# For a project mostly in C, the following would be a good se
# exclude patterns (uncomment them if you want to use them):
# *.[oa]
# *~

#Exclude intellij files from Git tracking
*.iml
*.idea

/rlocale/
/**.sem
```

2 Creating the Develop Branch in GitHub

By default when you create a GitHub project the **main** or **master branch** is created.
It is desirable that no one works in the **master branch**.
So a develop branch is created to code to go there for testing

3 Creating a Feature Branch in GitHub

It is desirable that no one works in the **master branch**.
Now GitHub is trying to be PC and calls it the "Main" Branch Master has a connotation to slavery.
Nor should someone work in the **develop branch**.
To overcome this it is customary to create Feature Branches. This is usually placed in /feature folder

3.1 Naming convention

There is no naming convention and every company has its own way of naming this branch.
Some use project Code(JIRA code) and some teams add the developers name or initials and some teams even add a short description of the change to be made.
It is a good practice to create a feature branch to do one change to the project

3.2 Pulling the changes into the develop branch

And to follow peer review process with the Repository owner who approves the

4 Creating a Release Branch in GitHub

It is desirable that no one works in the **release branch**.
This is used to hold the code going to the QA or UAT server
Each organization has its own practices
This is usually code ready for final test and sign off
Or approved code ready to go to Production on the go-live date

Code is pulled from the develop branch to the release branch
On the go-live day code is pulled from the release Branch to the Master Branch and deployed to the production server

Some organizations use continuous Development tools that automate the deployment to servers
Jenkins and Bamboo are such tools
Some organizations use SonarQube triggered by the above tools to validate the Java code quality.

Note that not all organizations follow this flow some companies have their own practices

5 Common Git commands used

5.1 Creating a new Git project from an empty Repository with existing code

You first create an empty project in github

Now you go to the root folder of your project using Git Bash

```
cd /xxx/yyy/zzz/MyProject
```

You create a local git Repository for you project

```
git init
```

Check the status of files in your local repository

```
git status
```

Adding your changes to your local repository

git add .   **or you can use:**   git add --all

Committing your initial changes (All your files) to your local repository

```
git commit -m "initial commit"
```

Setting up your GitHub Credentials

```
git config --global user.email "users_Email_Address@gmail.com"
```

```
git config --global user.name "Git_Hub_User_Name"
```

Linking the Remote repository to the local repository

```
git remote add origin https://github.com/nic0michael/antenna-plotter-microservice.git
```

The initial push to the master repository (After this you will never again push code to the master branch in the remote repository (GitHub)

```
git push -u origin master
```

5.2 Cloning an existing repository and checking out your feature to your workspace

```
git clone https://github.com/nic0michael/antenna-plotter-microservice.git
```

Display all the branches:

```
git branch -a
```

Checking out a specific branch

```
git checkout feature/XXXXXX_create_persistance_project
```

you are now ready to work with your feature's code

5.3 Pushing your code after making code changes

Check the status of files in your local repository

```
git status
```

Adding your changes to your local repository

```
git add .
```

Committing your changes to your local repository

```
git commit -m "your commit comment got here"
```

Pushing all committed changes in your local repository to the GitHub Server

```
git push origin feature/XXXXXX_create_persistance_project
```

5.4 Stashing or discarding your code changes

It often happens that two developers on working in the same branch on the same file
if you forget to pull every time before you make code changes you could create conflicts
sometimes you want to discard your changes and the pull the code changes made by the other developer

This is destructive so may want tomake folder level backups

```
git stash
```

5.5 Creating a (Feature)branch from the Develop Branch

Although you could do this with a git command GitHub provides an excellent GUI for doing that

You have two branches: **master and develop**
You want to create a "feature branch" from the develop branch.

```
git checkout -b myfeature dev
```

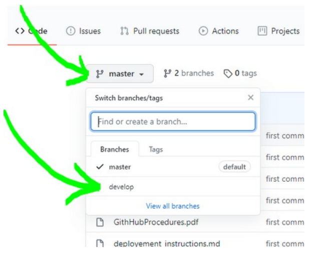
The danger with this is if you name your feature branch to an existing branch it will checkout that branch and not create a new feature branch

5.6 Setting the develop branch as the Default Branch using GitHub's GUI

Log int GitHub

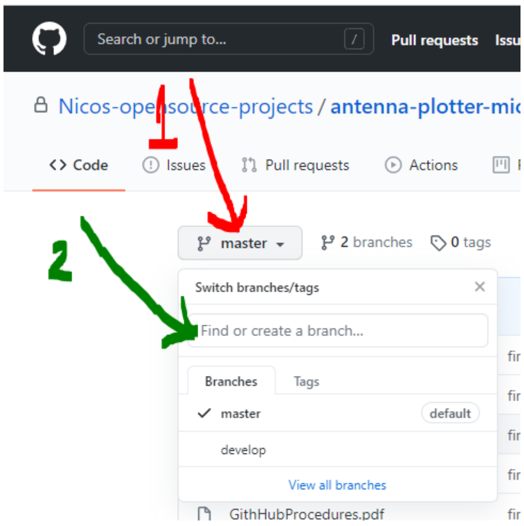
Open the GitHub Webpage for your Repository

Click on the Branch Dropdown List
Then Click on the Branch (develop) from which you wish to make a feature Branch if it is not the default branch make it the default branch

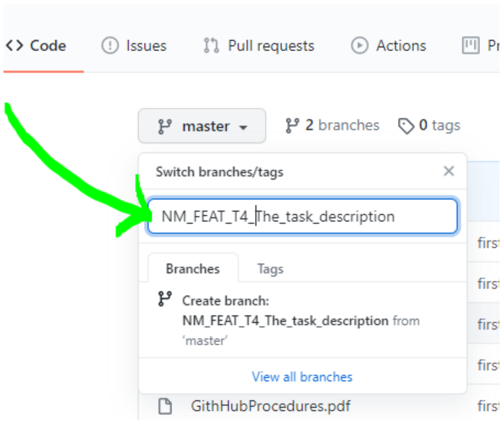


5.7 Creating a branch from the Develop Branch using GitHub's GUI

Step 1 click on Branch Dropdown List (by default the Develop branch should be the first entry)
Step 2 click in the textbox



Step 3 In the textbox type the new branch name



6 Git best practices

6.1 The Master Branch

This branch is never used to make changes and commit to.

It must always contain Clean Working Code

6.2 The Develop Branch

This branch is never used to make changes and commit to.

It must always contain Clean Working Code

The only to get new code here is by a pull request.

6.3 The Feature Branches

By creating feature branches from the Master Branch multiple developers can be able to make changes to the code in the master branch without polluting it

After committing the working tested code the developer will make a Pull Request to have the code Peer Reviewed and Feature Branch Merged into the Develop Branch

Should there be any conflicts the developer will make the fixes

The only branches a developer should checkout is from a Feature Branch and his/her feature branches

6.4 Pull Requests

This is the mechanism where after a developer has made changes in the developers Feature Branch the code will be reviewed by all the reviewers assigned to the pull request

When the reviewers are happy with the code they will complete the Pull process

If a reviewer is not happy with the code the reviewer can reject the pull request and advise the developer when code changes are needed

Initially the code from a Feature Branch will be pulled to the Develop Branch after a pull request and successful peer review

After new code has been pulled into the Develop Branch from a feature branch and after another pull request will be made to pull the code to the Master branch after another peer review

6.5 Release TAGS

These are made to mark the code in that point in time when it was ready for production.

6.6 Release Branches

Every time we are taking code to production for a Go-live that code will be from a Release Branch

In the POM file that version number will have the third digit set to zero and the word -SNAPSHOT removed.

RELEASE BRANCHES ARE NEVER DELETED

6.7 The Bugfix Branches

If a bug was found in Production after a go-live

a Bugfix branch should be made from the Release branch

The code fixed and the fixed code merged into the Release branch

The developer must also undertake to bring these changes to the Master Branch

6.8 Naming Conventions

6.8.1 Master Branch

This is generated by Bitbucket

6.8.2 Develop Branch

This is created from a new project from the Master Branch

6.8.3 Feature Branch

Name: feature/<JIRA Nr>_<Developers W Nr>_<A suitable description>

feature/266369-w5116880_change_monitor_persistance_project

6.8.4 Bugfix Branch

Name: bugfix/<JIRA Nr>_<Developers W Nr>_<A suitable description>

bugfix/266369-w5116880_fix_bugs_in_project

6.8.5 Release Branch

Name release/<JIRA Nr>_<A suitable description>_<golive date>

6.8.6 Release Tag

Name release/<JIRA Nr>_<A suitable description>