GitHub Documentation

(Second Draft: 07/08/2019)

Table of Contents

Introduction

The purpose of this document is to provide the necessary knowledge for the students to manage their code base effectively using the GitHub versioning application for their training code base.

This document is as comprehensive as it needs to be, there are Online Resources available in this page, and the student can also use Search engine tools and various online resources such as StackOverflow, etc.

The following are the topics in this document

- Pre-Requisites
- What is GitHub?
- Why use GitHub?
- Accessing the Cognixia repositories
- Overview of the Cognixia repositories structure
- Repository 101 for JUMP
- Terminology
- The Tools
- How-To
- Workflow
- GitHub Access
- Online Resources

This document assumes the developer has no knowledge of Git.

What this document doesn't do is compare Git with other type of applications which provides similar function.

Pre-Requisites

1) Installation of Git Bash command prompt application
2) Login Access to GitHub via the student's account
3) Permissions to access various repositories

What is GitHub?

GitHub is a commercial product from Microsoft. This is an external resource available on the web. It can be access via https://www.github.com/jumpjune2019

GitHub is a Version Control System (VCS).

GitHub is in fact a Distributed Version Control System (DVCS). Every 'snapshot' of a repository is in effect a full backup of the repository at that point in time. All files and folders which make up the repository are 'versioned' every time updates are performed between the local repository and the one cloned from GitHub.

While you are in your local repository, you can manage your codebase using branches. Each branch is a copy from another branch. In GitHub there is always a 'master' branch.

You are able to push any branches from your local repository to your GitHub repository.

Branches are often used as a way of managing your code base, you will create a branch with the purpose of working a specific part of the application for specific reasons. When you are satisfied with your changes, you can merge back to another branch, often, your 'master' branch.

Why Use GitHub?

GitHub is meant for sharing a repository to multiple collaborators. It is a collaborative environment.

Each collaborator is able to 'clone' the repository to their local PC. Code pushes from local repository to GitHub repository are 'merged' to a branch, typically the 'master' branch if it involves a continuous integration process.

In a team environment we commonly see 2 types of environment setup

1) Collaborators clone from the original repository to their local PC, or;
2) Collaborators create a "Fork" of the original repository. A forked repository also resides in GitHub. The collaborator will clone the forked repo to their local PC.

For the training session at JUMP, there will be a mix of the two techniques.

Each student will be able to push code to their forked repo, then create a 'pull request' to have their updates merged with the original repo or just clone from JumpJune2019.

Accessing the JUMP Repositories

All Repositories are managed under the JumpJune2019 account.

Any user can have their own personal repositories as well, for the most part, 'forked' repositories are owned by the user or they can clone directly from JumpJune2019 as well.

Repository	**Role**	**Comments**
Core_Material	Main Repo	This is where students can find the files for the training and they can clone this repo as opposed to forking it, because they will never be able to upload to it, they only have 'read' only access.

Terminology

- Repository
- Fork
- Clone
- Branch
- ‘master’ branch
- Checkout
- Commit
- Push
- Pull
- Snapshot
- Merge
- Hash
- .gitignore

Repository	A GitHub entity which holds the entire content of a project (web app, documentation, etc., basically, computer files and folders). Updates to this content is ‘versioned’ as a ‘snapshot’ of the entire content at any given point in time updates are performed.
Fork	A GitHub function to create a repository from another repository within the GitHub environment.
Clone	A function which allows the creation of a copy of a repository to a local PC.
Branch	Within either the local or the repository cloned from, branches are snapshot of the code base. There is always a ‘master’ branch. Branches are created with the purpose of developing code in a targeted manner. Branches can be merged.
‘master’ branch	The default branch all repositories have. Locally, developers should never ‘develop’ solution in the ‘master’ branch. They should however, merge to the local ‘branch’ once they have tested their solution.
Checkout	A function on the local system to either 1) switch between branch or 2) create a new branch
Commit	Developers can choose the files they wish to ‘commit’ so that they can be pushed from their local repository to their GitHub repository. Once files are ‘added’, then a ‘commit’ command is issues. This command always involves the writing of a message which describes the nature of the commit operation
Push	After a commit command has been issued, the developer then invokes a push command. This command will push all the files added and confirmed from the commit command to the GitHub repository which is linked to the developer’s local repository
Pull	Retrieving code from the online repo to your local repo, a git pull retrieves data and merges your code base.
Snapshot	A term used when a push command has been effectively done. Basically, a snapshot represents an entire set of a repository for all branches, which can be restored or reviewed using command line tools or the GitHub web user interface, a snapshot is a full repository at that point in time it was taken and can be restored at any given time.
Merge	Merge is a command which is done on a local system when merging code from 2 branches, or when your code base is not in sync with the GitHub repository. Typically, you get to parse a file and determine which part of the code you wish to merge. Depending on the complexity of the merge, it may become a manual operation, although, with some forethought, it can be done automatically, but it does assume some diligence on the part of the developer.
Hash	Each commit/push operation assigned a hash value to a collection of files being committed.
.gitignore	List of files or file patterns which never gets tracked thus never gets pushed to a repo. This file is always located at the root of a repository

Local Repository Tools

You will have 2 tools at your disposal in order to manage your GitHub repository.

- Git Bash
- IDEs

Note: there are certain operations which can only be done using Git Bash

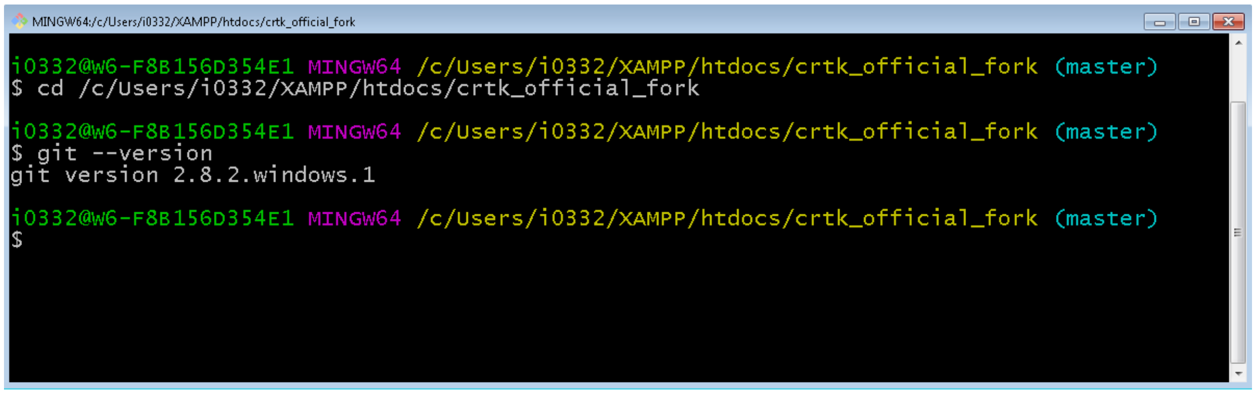
Git Bash

Git Bash comes in 2 flavors for OFI developers.

- Git Bash command prompt
- Windows command prompt with Git support

Git Bash command prompt

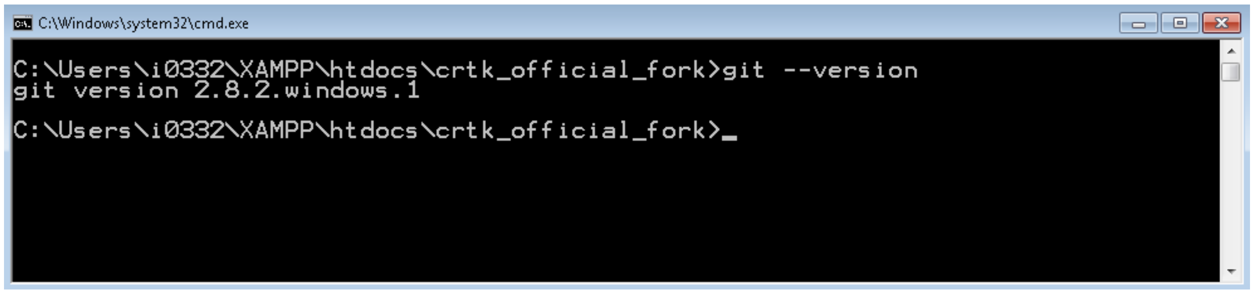
This version is a Linux Shell which has full access to your local PC and all it's resources (shared and network drives). You will use Unix syntax when working with this shell. It is recommended to use this, especially since most of the online information available on Git is based on Linux syntax.



```
i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (master)
$ cd /c/Users/i0332/XAMPP/htdocs/crtk_official_fork

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (master)
$ git --version
git version 2.8.2.windows.1

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (master)
$
```

Windows command prompt with Git support

This is your windows command prompt which uses DOS syntax and allows you to run Git commands.



```
C:\Users\i0332\XAMPP\htdocs\crtk_official_fork>git --version
git version 2.8.2.windows.1

C:\Users\i0332\XAMPP\htdocs\crtk_official_fork>_
```

How-To

- Fork a repository
- Clone a repository
- Create a branch
- Switch between branches (local)
- View all available branches in local repository
- View all available branches in online repository
- Push a local branch to online repository
- Add file(s) to push to online repository
- Remove file(s) before commit
- Remove file(s) after commit
- Commit file(s)
- Amend a commit before push
- Push files to online repository
- Verify online repository for updated content
- Updating the local repository with content from the online repository
- Merge local branches
- Deleting a branch
- Deleting a repository
- Git Help
- Setup Chrome to Open Git Help
- Avoid Merge Conflicts
- Setup a .gitignore file

Fork a repository

Using the GitHub interface

1. Select a repository
2. Select the fork command
3. Provide a project and repository name
4. Click the "Fork Repository" button

Clone a Repository

Using the GitHub

1. Select a repository
2. Click on the Clone command
3. Copy the URI from the text field to your Windows clipboard (ctrl-c)

Using Git Bash

4. Open a Git Bash session
5. Change to the drive/folder where your wish the clone the repository, this will become your local repository (i.e. c:/users/i0332/)
6. Type: **`git clone`** and **add a space**
7. Paste the command (Shift-Insert) when using Git Bash (linux) and **Press Enter**

This will create a new folder.

Create a Branch

There are many ways to create branches, we will use the Git Bash command line interface and Linux syntax. It is best for developers to create branches locally and push them to their repository, if necessary.

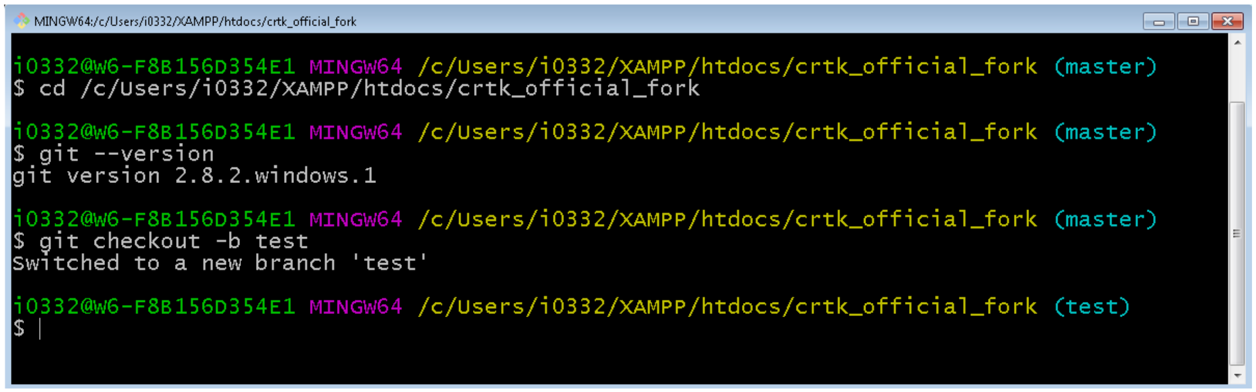
Note: first time you create a branch in a new local repository, you will always be in a 'master' branch. But after that, you can always create new branches from other branches, try and use a naming convention which allows you to understand the relationship between branches.

Scenario: create a branch named 'test' from the 'master' branch

Using Git Bash

1. change to the local repository directory
2. type: **`git checkout –b test`** and **Press Enter**
3. a new branch will be created name 'test' in your local repository

Your command prompt should look something like this:



```
i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (master)
$ cd /c/Users/i0332/XAMPP/htdocs/crtk_official_fork

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (master)
$ git --version
git version 2.8.2.windows.1

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (master)
$ git checkout -b test
Switched to a new branch 'test'

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$ |
```

Switch between branches (Local)

Scenario: switch from the ‘test’ branch to the ‘master’ branch

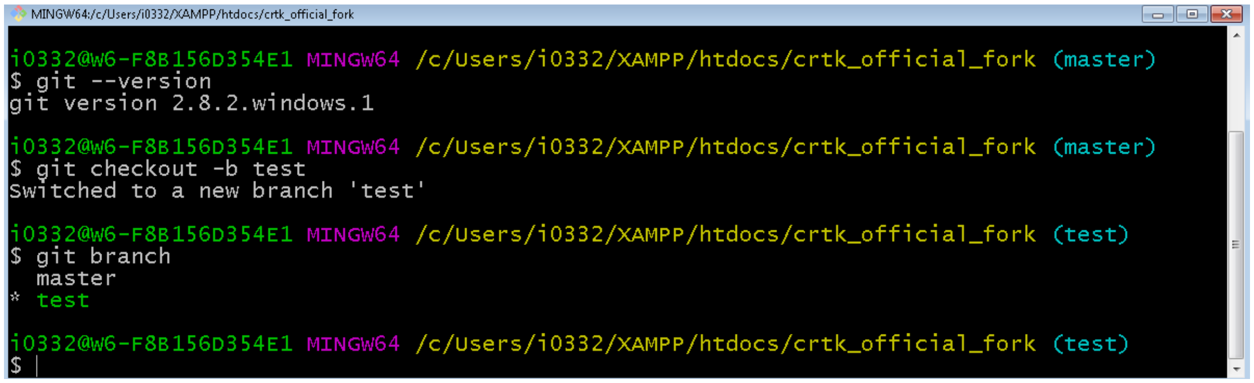
Using Git Bash

1. change to the local repository
2. type: **`git checkout master`** and **Press Enter**
3. you will have changed to the ‘master branch

View all available branches in local repository

Using Git Bash

1. Change to the local repository
2. Type: **`git branch`** and **Press Enter**
3. A list of branches will be displayed, the current branch will be displayed with an asterisk* character



```
i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (master)
$ git --version
git version 2.8.2.windows.1

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (master)
$ git checkout -b test
Switched to a new branch 'test'

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$ git branch
  master
* test

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$ |
```

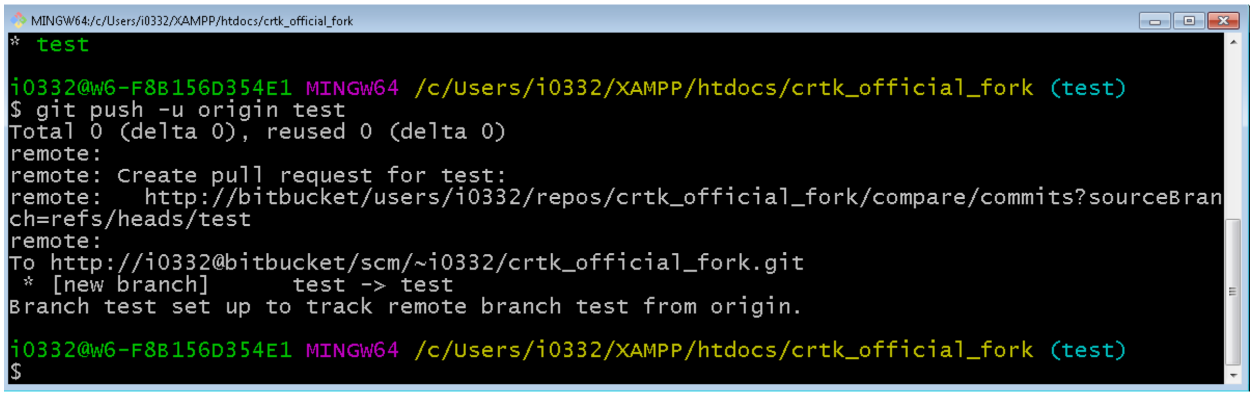
Push a local branch to online repository

Scenario: Pushing the ‘test’ branch to online repository

Using Git Bash

1. Type: **`git push –u origin test`** and **Press Enter**

You can verify using GitHub that the 'test' branch has been pushed.



```
* test

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$ git push -u origin test
Total 0 (delta 0), reused 0 (delta 0)
remote:
remote: Create pull request for test:
remote:   http://bitbucket/users/i0332/repos/crtk_official_fork/compare/commits?sourceBran
ch=refs/heads/test
remote:
To http://i0332@bitbucket/scm/~i0332/crtk_official_fork.git
 * [new branch]      test -> test
Branch test set up to track remote branch test from origin.

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$
```

Add file(s) to push to online repository

You will add/remote/modify files from your project which are located on your local PCs.

Git will track those modification, developers can decide at any given time when to push code to the online repository and which file(s) to add.

Using the Git Bash tool, the command always starts with **`git add <file(s)>`**.

You will be using your Git Bash' operating system syntax to refer to file(s).

Examples are:

- $ **`git add *`**
- $ **`git add -all`**
- $ **`git add /apps/crtk/view/Main.js`**
- $ **`git add /apps/crtk/controller/*`**

If you need to add 2 distinctive files, you can just type the **`git add`** command sequentially, each statement refers to a file.

Adding file(s) is the first step towards pushing code to the online repository.

The second step is to execute a 'commit' command.

The third step is to push the code by executing a 'push' command.

The command **`git status`** will always provide with information to the files which are being added as well as the status of your local repository at that point in time.

If you wish to view the list of file(s) which are queued, use the **`git status`** command.

Remove file(s) before commit

If you need to remove some file(s) from being committed to the repository, with the Git Bash tool, the command is **`git rm 'file(s)'`**.

You will be using your Git Bash' operating system syntax to refer to file(s).

Examples are:

- $ `git rm /apps/crtk/view/Main.js`
- $ `git rm /apps/crtk/controller/*`

If you need to add 2 distinctive files, you can just type the **`git rm`** command sequentially, each statement refers to a file.

Use the **`git status`** command to check which files are staged to be committed. The files you have removed should no longer be listed.

Remove file(s) after commit

If you have executed a **`git commit`** command only to realize that some of the files which you have staged to be pushed should not be commited, then you would need to follow these steps

1. Remove files from being committed
2. Amend your commit command which will now only tag the files you wish to commit

To remove some file(s) from being committed to the repository, with the Git Bash tool, the command is **`git rm <file(s)>`**.

You will be using your Git Bash' operating system syntax to refer to file(s).

Examples are:

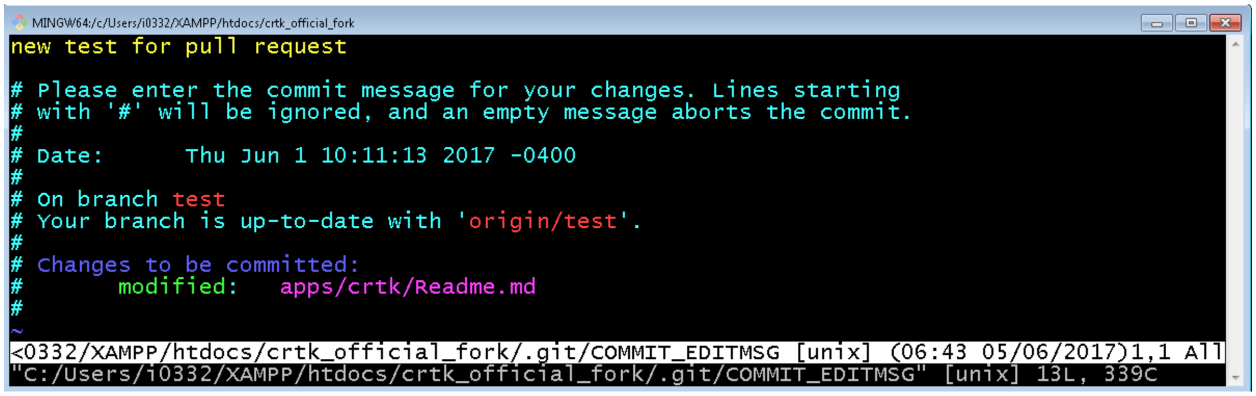
- $ `git rm /apps/crtk/view/Main.js`
- $ `git rm /apps/crtk/controller/*`

If you need to add 2 distinctive files, you can just type the **`git rm`** command sequentially, each statement refers to a file.

To amend your commit, you will type the following:

- $ `git commit --amend`

You will get the following:


MINGW64:/c/Users/i0332/XAMPP/htdocs/crtk_official_fork

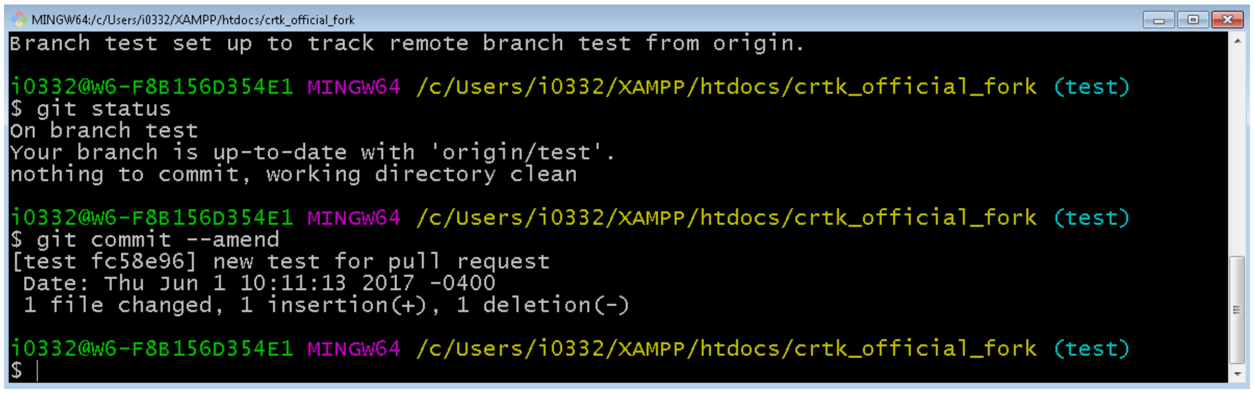
```
new test for pull request

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Jun 1 10:11:13 2017 -0400
#
# On branch test
# Your branch is up-to-date with 'origin/test'.
#
# Changes to be committed:
#       modified:   apps/crtk/Readme.md
#
~
<0332/XAMPP/htdocs/crtk_official_fork/.git/COMMIT_EDITMSG [unix] (06:43 05/06/2017)1,1 All
"C:/Users/i0332/XAMPP/htdocs/crtk_official_fork/.git/COMMIT_EDITMSG" [unix] 13L, 339C
```

The **`commit --amend`** command will open in a "VI" editor.

In the scenario where you have removed some files from being committed, you might wish to keep the same message, in this case, you only need to save the commit.

You can of course, modify the commit message to reflect a message more suitable to explain the nature of the commit.

To save your changes or keep the same message, **Press the Escape key** 'Esc', then **type** **`:wq`** and **Press Enter**



MINGW64:/c/Users/i0332/XAMPP/htdocs/crtk_official_fork

```
Branch test set up to track remote branch test from origin.

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$ git status
On branch test
Your branch is up-to-date with 'origin/test'.
nothing to commit, working directory clean

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$ git commit --amend
[test fc58e96] new test for pull request
 Date: Thu Jun 1 10:11:13 2017 -0400
 1 file changed, 1 insertion(+), 1 deletion(-)

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$ |
```

Commit file(s)

The purpose of each commit is to link a hash value to the files which are going to be pushed to the online repository.

This value is automatically generated.

There are 3 stages required to push code to the repository

1. Add file(s)
2. Commit
3. Push

In the commit stage, after you have added file(s), all you need to do is this:

- $ **`git commit –m "some message about this commit"`**

Note: Use double-quotes for your commit message.

Amend a commit before push

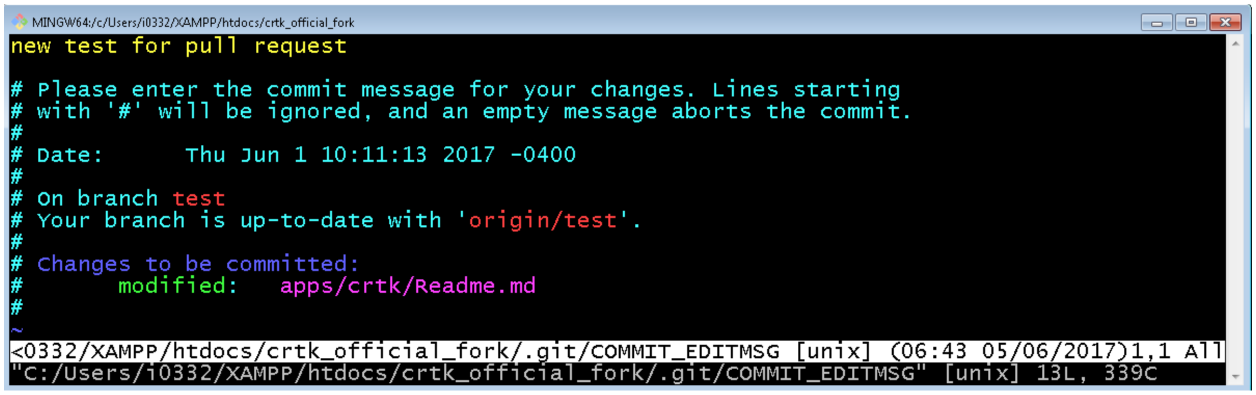
There are at least 2 reasons why you would wish to amend a commit before pushing code to the online repository:

1. The message is wrong
2. You need to modify the list of files being committed (removing, adding new, modified current files and re-added them)

To amend your commit, you will type the following:

- $ **`git commit --amend`**

You will get the following:



MINGW64:/c/Users/i0332/XAMPP/htdocs/crtk_official_fork

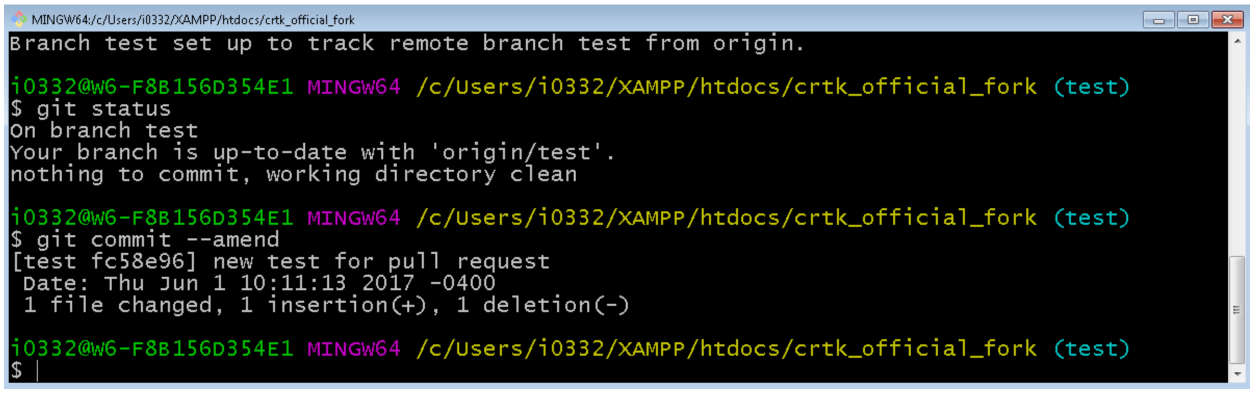
```
new test for pull request

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Thu Jun 1 10:11:13 2017 -0400
#
# On branch test
# Your branch is up-to-date with 'origin/test'.
#
# Changes to be committed:
#       modified:   apps/crtk/Readme.md
#
~
<0332/XAMPP/htdocs/crtk_official_fork/.git/COMMIT_EDITMSG [unix] (06:43 05/06/2017)1,1 All
"C:/Users/i0332/XAMPP/htdocs/crtk_official_fork/.git/COMMIT_EDITMSG" [unix] 13L, 339C
```

The **`commit --amend`** command will open in a "VI" editor.

The message can be modified, or you may wish to keep the same message.

Either way, you will need to save the commit.

To save your changes or keep the same message, **Press the Escape key** 'Esc', then **`type :wq`** and **Press Enter**



MINGW64:/c/Users/i0332/XAMPP/htdocs/crtk_official_fork

```
Branch test set up to track remote branch test from origin.

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$ git status
On branch test
Your branch is up-to-date with 'origin/test'.
nothing to commit, working directory clean

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$ git commit --amend
[test fc58e96] new test for pull request
 Date: Thu Jun 1 10:11:13 2017 -0400
 1 file changed, 1 insertion(+), 1 deletion(-)

i0332@W6-F8B156D354E1 MINGW64 /c/Users/i0332/XAMPP/htdocs/crtk_official_fork (test)
$ |
```

Push files to online repository

There are 3 stages required to push files to a repository

1. Add files
2. Commit
3. Push

The most basic syntax for pushing is **`git push.`**

The –u parameter instructs the command to create a new branch online.

A more comprehensive syntax is as follows:

- $ **`git push <parameters> <repository> <branch>`**

Note: You can choose a repository to push to, although the alias for your current repository is 'origin'.

Using Git Bash and assuming the branch is available in the online repository

1. Change to the local repository
2. Ensure you are in the correct branch
3. Type: **`git push`** and **Press Enter**

Using Git Bash and assuming the branch has not be created in the online repository

1. Change to the local repository
2. Ensure you are in the correct branch
3. Type: **`git push –u origin <name of the branch>`** and **Press Enter**
   `i.e.` **`git push –u origin test`**

Updating the local repository with content from the online repository

The command to update your local repository is **`git pull`**.

Scenarios:

- Update all local branches
- Update a specific branch

Update all local branches

- $ **`git pull --all`**

Update a specific branch

1. Switch to the local branch you wish to update from the remote repository
2. $ **`git pull`**

Note: **`git pull`** performs two operations for you, it will **fetch** data from the remote and **merge** data with your local changes. You might run into merge conflicts.

Merge local branches

Using Git Bash

Scenario: merging the content of Branch B into Branch A.

1. $ **`git checkout <BranchA>`**
2. $ **`git merge <BranchB>`**

Delete a Branch

Using Git Bash

- Delete a local branch
- Delete a Remove Branch

Delete a local branch

- `$ git branch -d <branch_name>`

Delete a Repository

On your local PC, a repository is a folder, so, as long as you have no applications which have file(s) or folder(s) opened from the repository (folder), you can simply delete the folder.

In GitHub, you delete a repository via its settings page.

Git Help

In Git Bash, you can easily have access to help information on any Git command.

For example, to get information on the **git commit** command you would type:

- `$ git help commit`

Avoid Merge Conflicts

There are many topics on merge conflicts online.

Core reasons for merge issues are:

1) Your local repository is behind in terms of commits against the online repository
2) Same file(s) are modified by 2 or more developers and are not in sync

Note: The above list is not exhaustive.

Articles on Merging

https://www.atlassian.com/git/articles/git-team-workflows-merge-or-rebase
http://kernowsoul.com/blog/2012/06/20/4-ways-to-avoid-merge-commits-in-git/
https://imagej.net/Git_Conflicts
https://randyfay.com/content/avoiding-git-disasters-gory-story

Setup a .gitignore file

Before the repo is initialized

1) Create a file at the root of your repository and name it .ignore
2) Add files and file patterns as required

3) For example:

```
.idea/
app.config.json
.architect
.project
.sass-cache/
ext/
ext651/
touch/
temp/
build/
*sencha*.log
apps/Pdap/classic.json
apps/Pdap/classic.jsonp
```

After a repo has been initialized / updating a .gitignore file with new content

1) Update the content of your .gitignore file
2) At the command prompt remove all tracked file using **`git rm -r --cached .`**
3) **`git add .`** to add files back except those listed in the .gitignore file
4) **`git commit -m`** `“fixed untracked files..”`
5) **`git push`**

Working from other branches

There are times where you might wish to create a branch from something other than the ‘master’ branch.

Remember if you do this that when you merge, merge to the branch you originated.

For example:

- Master
- FixLayoutIssue

These are 2 branches.

If you need to do something from FixLayoutIssue, you could create a branch and name it “FixLayoutIssue_Test”

When you are done working with FixLayoutIssue_Test, you would merge its content into the FixLayoutIssue branch, then merge that branch to your master branch.

Use a naming convention which allows you to easily understand the relationship between branches, assuming ‘master’ is always your ‘root’ branch, because.. It is! ☺

Don’t forget to use **git branch** to list all your branches in the local repository.

Online Resources

- https://help.github.com/articles/git-and-github-learning-resources/
- https://www.atlassian.com/git
- https://www.atlassian.com/git/tutorials
- https://www.tutorialspoint.com/git/
- https://git-scm.com/docs/gittutorial