# GIT CHEAT SHEET

## After checking out a master branch always **PULL** before doing anything else

## Starting work on an existing file

1. Checkout local master branch
2. Pull on master branch to get new commits in remote repository
3. Create and checkout new local branch
4. Push new local branch to remote repository
5. Make changes
6. Test changes
7. Add untracked changes
8. Commit changes
9. Push local branch to remote repository
10. Checkout local master
11. Pull on master
12. Merge new local branch to local master
13. Resolve any conflicts
14. Push changes on local master to remote repository
15. FTP local master changes to live
16. If finished with new branch delete it in local and remote repository

# GIT COMMAND LINE INTERFACE INSTRUCTIONS

# Open Git Bash Command Line Interface

1. Open the command line interface Git Bash   
   (*click the windows button, type ‘git’ and select* ***Git Bash****)*
2. Cd to the working directory:  
   $ cd c:/<your directory path>

# Getting a remote repository

1. Cd to the directory you wish to work in.
2. If it is a new directory initialise git in the new directory   
   $ git init
3. Clone the remote repository, for example a repository called ‘test’:  
   $ clone http://tor-gitlab/torfx-web/www.torfx.com.git/test/test  
   This creates a directory called test, initialises a .git directory inside it, pulls down all the data for that repository and checks out a working copy of the latest version of test.

# Checkout branch

1. To checkout a branch type:

$ git checkout <<branch name>>

1. If you are working in one branch and need to checkout another you may be prevented from doing so because you have not committed changes. To overcome this without having to commit half completed work you can temporary ‘stash’ changes.  
   See [Stash Changes](#_Stash_Changes).
2. To see a list of branches type:  
   $ git branch

# Stash changes

Stash should only be used as a temporary measure. It is a good idea to delete stashes when no longer required.

1. To temporarily save changes to give a clean branch to enable you to checkout another branch type:

$ git stash

1. To re-apply the last stash type:  
   $ git stash apply
2. To view a list of stashes type:  
   $ git stash list
3. To apply the last stash type:  
   $ git stash apply
4. To apply a particular stash type:  
   $ git stash apply --<name of stash>
5. To clear stashes from a branch type:  
   $ git stash clear

# Checking the status of your files

1. Cd to the working directory and type:  
   $ git status
2. Changes not staged for commit and untracked files appear in red.
3. Changes staged for commit appear in green.
4. If your changes are committed but not pushed you will see:  
   On branch <branch name>  
   Your branch is ahead of 'origin/<branch name>' by 1 commit.  
    (use "git push" to publish your local commits)  
   nothing to commit, working tree clean
5. If all is up to date you will see:

On branch <branch name>

Your branch is up to date with 'origin/<branch name>'.

nothing to commit, working tree clean

# Tracking new files

1. You begin tracking new files with the $ git add command.
2. To track individual files type:  
   $ git add <<file name>>
3. To track more than one file add the filenames spaced with a comma:  
   $ git add <<file name>>, <<file name>>
4. To track all untracked files type:  
   $ git add .

# Committing files

1. Files must be tracked (added) before you can make a commit.
2. When committing files you should always include a meaningful message to aid in recognising changes at a later date.
3. To commit tracked files type:  
   $ git commit –m “meaningful message”

# Pushing files

1. Use the push command to push commits made on your local branch to a remote repository:  
   $ git push

# Creating a new local branch and push branch to remote repository

1. Checkout master branch.  
   $ git checkout master
2. Pull on master branch to incorporate any new commits.  
   $ git pull
3. Create new local branch (use meaningful name)  
   $ git branch <branch name>
4. Push new local branch to remote repository:  
   $ git push --set-upstream origin <branch name>
5. Start work on files in the branch.

# Merging branches with master branch

1. Finish work on a branch and commit and push changes to remote repository.
2. Checkout master branch:  
   $ git checkout master
3. Pull on master branch to incorporate any new commits.  
   $ git pull
4. Merge master branch with branch  
   $ git merge <branch name>
5. If there are any conflicts you will see something like:  
   $ git merge <branch name>  
   Auto-merging <file name>  
   CONFLICT (content): Merge conflict in <file name>  
   Automatic merge failed; fix conflicts and then commit the result.  
     
   Git hasn’t automatically created a new merge commit. It has paused the process while you resolve the conflict. If you want to see which files are unmerged at any point after a merge conflict, type:  
   $ git status  
   You will see something like:  
   $ git status  
   On branch master  
   You have unmerged paths.  
    (fix conflicts and run "git commit")  
     
   Unmerged paths:  
    (use "git add <file>..." to mark resolution)  
     
    both modified: index.html  
     
   no changes added to commit (use "git add" and/or "git commit -a")  
     
   To resolve conflicts see [Resolve Merge Conflicts](#_Resolve_Merge_Conflicts).
6. Push master branch to remote repository:  
   $ git push

# Exit git process

1. There may be times when you need to exit a git process that is waiting for an action to finish. To exit a git process hold down the CTRL key and type C.

# Resolve Merge Conflicts

1. There are many ways of resolving conflicts. The easiest is probably to open the problem file in your text editor. Conflicts are marked by:  
   <<<<<<< HEAD  
   <div id="footer">contact : email.support@github.com</div>  
   =======  
   <div id="footer">  
    please contact us at [support@github.com](mailto:support@github.com)  
   </div>  
   >>>>>>>   
     
   <<<<<<< HEAD marks the start of the conflict in the master branch.  
   ======= divides the master and branch changes.  
   >>>>>>> marks the end of the conflict in the branch.
2. Delete or edit the changes you require and save the file.
3. In Git Bash type:  
   $ git add

# Delete local and remote branches

1. When a branch is no longer needed it is good practice to delete it.
2. Checkout the master branch:  
   $ git checkout master
3. To delete a branch locally type:  
   $ git branch –d <branch name>
4. To delete a branch in the remote repository type:  
   $ push origin – delete <branch name>

# Checkout a remote branch

1. Fetch the origin repository:  
   $ git fetch origin
2. Checkout the required branch  
   $ git checkout <branch name>  
    *In older versions of git you may have to type:* $ git checkout –track origin/<branch name>

# BASIC GIT COMMANDS

Get an existing from the server for the first time  
$ git clone git@example.com:repositoryname

See what's changed  
$ git status

Commit  
$ git commit -m "good description"

Push local commits to the server  
$ git push

Get and merge updates from the server  
$ git pull

Stage a file for the next local commit  
$ git add <file name>

Stage all files for the next local commit  
$ git add .

Create a new local branch and check it out  
$ git checkout -b <branch name>

Switch to a local branch  
$ git checkout <branch name>

Switch to the previous branch  
$ git checkout -

List local branches  
$ git branch

Delete a local branch  
$ git branch -d <branch name>

Insert changes on the server before your local changes  
$ git pull --rebase

Temporarily discard local changes  
$ git stash

Re-apply stashed away changes  
$ git stash apply

Re-apply stashed away changes an delete them from stack  
$ git stash pop

Find out who wrote something  
$ git blame <file name>

Restore a deleted file (notice the space before and after the double dash)  
$ git checkout -- <file name>