Introduction

This is a basic guide to using Git for T3D contribution and development.

This guide assumes you already have installed git and it is integrated into your console. To test that this is the case, open up a command-line window (windows key + r -> type in "cmd" without the quotes -> press enter) and write

```
git
```

And then press enter, if it says the command was not recognized, then you probably haven't set Git up properly.

Platform

This tutorial is aimed towards Windows developers, but I don't see why it wouldn't work for Linux or Mac as well, aside from minor differences in terminology.

Getting the code

Cloning vs. Forking

When should you fork and when should you clone?

Cloning

Cloning is when you need a local copy of a remote repository. You should not clone if your intention is to push your changes to a remote repository.

Forking

Forking is for when you need to create copy of the repository on GitHub, which you can make changes to and merge those changes in to the official repository.
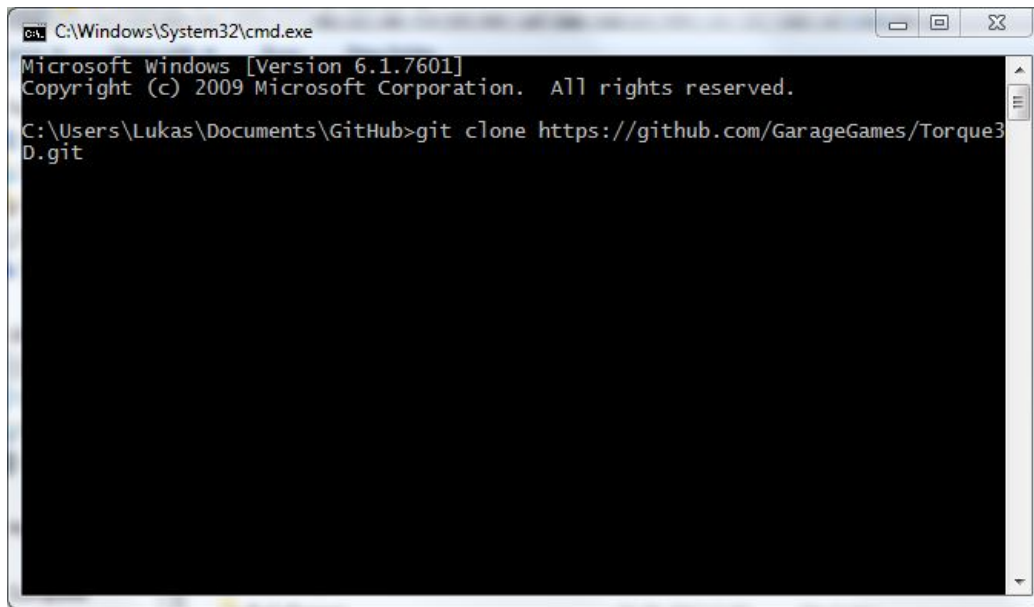
Cloning

Cloning a repository is really easy. First open a command-line window, and navigate to the parent repository of where you want your repository to be downloaded to. (e.g. C:/Users/Lukas/My Documents/GitHub).
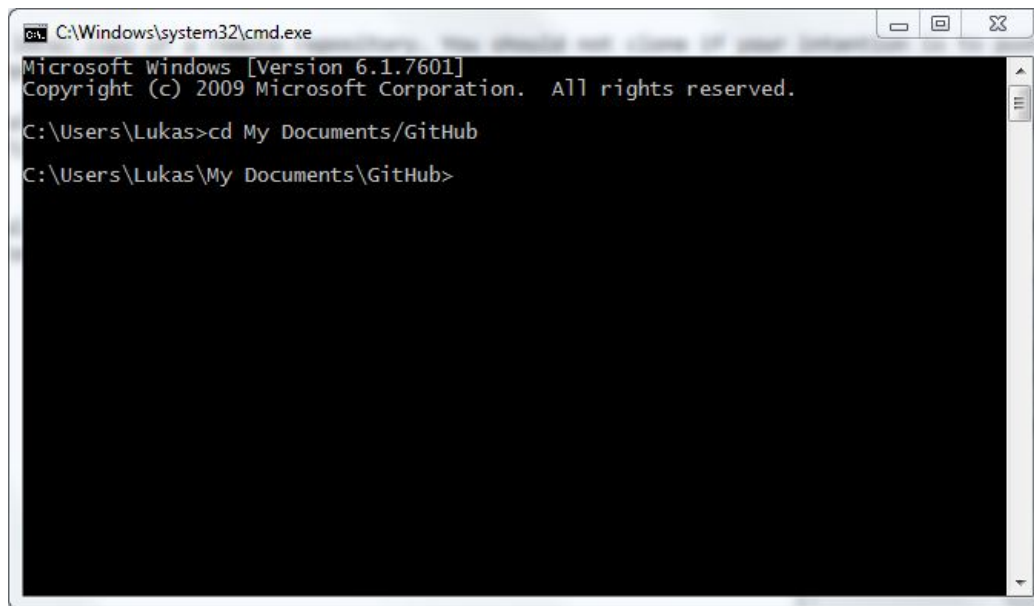
Then you write: `git clone repositoryURL` where repositoryURL is the URL for the .git repository on GitHub, for Torque3D it is:

```
git clone https://github.com/GarageGames/Torque3D.git
```

The command in the window looks like this:

```
C:\Windows\System32\cmd.exe

Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Lukas\Documents\GitHub>git clone https://github.com/GarageGames/Torque3
D.git
```

Just press enter, and you are done! It will now clone the repository into a new folder called "Torque3D".

Forking

Forking a repository is simple as well. When you want to fork a repository, you do it over the GitHub website.
Simply go to https://github.com/GarageGames/Torque3D and press the "fork" button.

And now just clone your forked repository, which you can find at *github.com/(your GitHub username)/Torque3D*
For example mine is located at https://github.com/lukaspj/Torque3D.

To get a local copy of your repository you will have to clone it as explained earlier, except this time you shouldn't use the URL for the Torque3D repository but instead the URL for your own repository, which you can find by going to your repository on GitHub.com and find the box that says: "**HTTPS** clone URL" and copy the link inside that box.
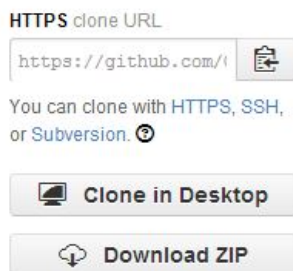
For me it is:

```
https://github.com/lukaspj/Torque3D.git
```

Contributing

If you want to contribute bug-fixes or other code to the T3D repository, then you should be familiar with the procedure for doing this.
The general "Contribution" workflow I use is whenever I there is something I want to change (e.g. a bug-fix) is:

1. Create a new branch for the change
2. Checkout that branch
3. Make the changes

4. Add the changes
5. Commit the changes
6. Push the changes to my own repo on GitHub
7. PullRequest the changes to the official repo

So lets go through this procedure step-by-step.

Branches

Branches are basically different versions of the same base code. When you create a new branch, you can edit it all you want, then checkout master and nothing will have changed.
It is good practice to create a branch for even the smallest change you make, this makes it easy to track, and undo changes if an issue arises later on.

Creating a branch
Creating a branch is very simple, just write

```
git branch new_branchname
```

Where new_branchname should be replaced by the name of the new branch you want to create.

Changing the current branch
When you want to change the branch you are currently working on you use the `checkout` command, it's very simple:

```
git checkout branchname
```

Where branchname should be replaced by the name of the branch you want to start working on.

Upstream
We need to tell our new branch where it should push any commits you've made to. We do this by using the `--set-upstream` modifier on the push command:

```
git push --set-upstream origin remote_branchname
```

Where remote_branchname should be replaced by the branch on the remote repository you want to push to (typically this will be the same branchname as your current branch, if you use the same branch name then GitHub will usually create that branch for you automatically.

Now you just need to make the changes to the code! Fix the bug, add the feature, whatever your intention with the branch was.

Committing

When you are done modifying the code, you need to *commit* the changes before pushing them.
First you need to *add* the files you want to commit, you do this with the `git add` command:

```
git add newfile.txt
```

The *add* command uses wildcards so you don't have to type in every file you want committed:

```
git add * // Adds everything
git add *.txt // Adds all .txt files
git add *foo.txt // Adds all files ending with foo.txt
git add foo.txt // Adds foo.txt
```

Committing
Now that you have added the files, you can now commit them by using the `commit` command.

```
git commit
```

This will add all the files you have changed to a new commit, you can use the `-m` modifier if you only have a short commit message which you can write directly in the console.

```
git commit -m "Commit message"
```

Pushing

Now that you have committed your changes, you should now push them to the online repository. For this you will use the `push` command:

```
git push
```

It may ask you for a username and password, for this just use the same username and password as you use for GitHub.com

Summary

git branch NewFeature
git checkout NewFeature
git push --set-upstream origin NewFeature
// Make the changes
git add *
git commit -m "Wow new feature that does this"
git push

Pullrequests

Assuming you have successfully pushed the commit to the online repository, you should now be able to do a pullrequest. Pullrequests are done through the website, and are very simple to perform.
Go to your online repository and press the green `Compare and review` button.
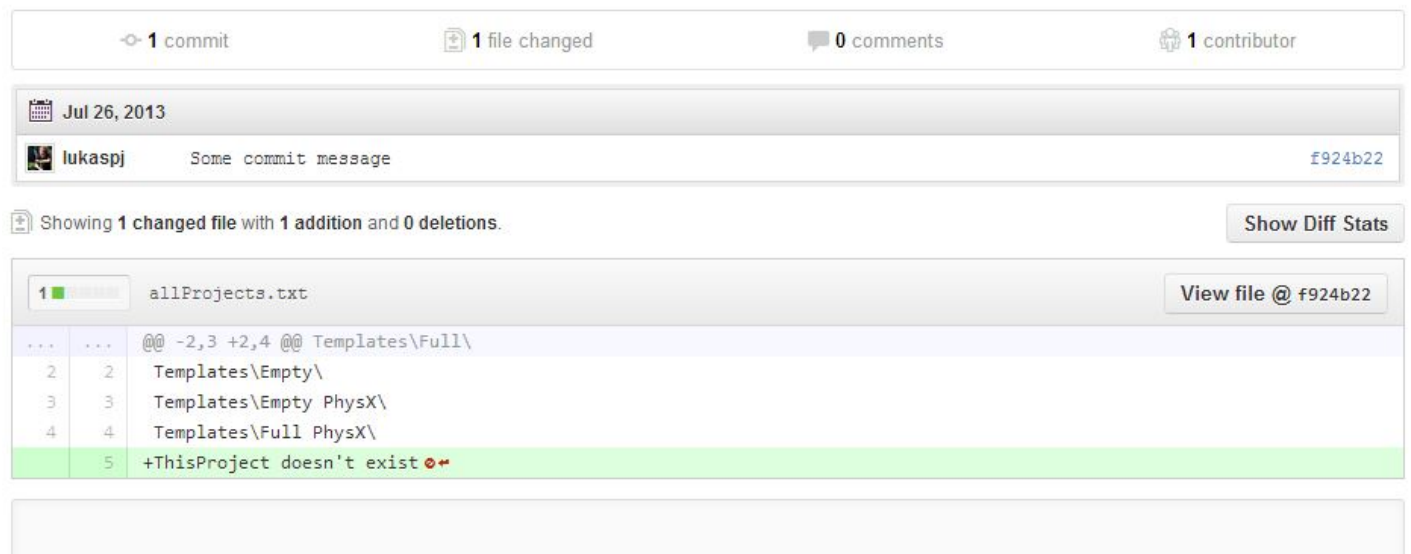

Now you come to the *compare page*, on this page you can compare your repository with the official one and create pullrequests from a branch in your repository to a branch in the official one.

You can edit what branches and repositories you want to compare by clicking the *"Edit"* button
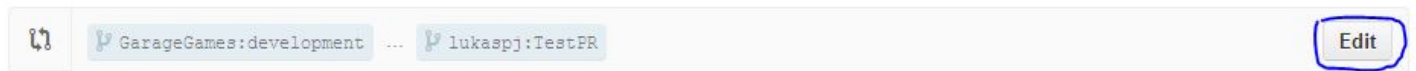
And you can see what changes has been made below that bar:

Now to actually make the PullRequest, click on the large button that says *"Click to create a pull request for this comparison"*

The dialog that then pops up will feature a title and a description. Here you should describe all the commits that your pull request involves! (If you only have 1 commit, copy-pasting the information from that commit will be fine) along with a "how to use" (or "how to replicate the bug this PullRequest fixes"). Then you just press *"Send pull request"* and thats it! Now you just have to wait untill the PullRequest is accepted/rejected and act accordingly. Be sure to be ready to help out if there is some issues with the PullRequest.
Make sure to checkout the guidelines below.

Important guidelines for contributing

Always pullrequest to the development branch
New changes should always go into the development branch before they go into the master branch, this is because they need to be QA'd before going into the Master branch. If you don't pullrequest to the development branch, your pullrequest will be rejected.

Avoid redundancy
If you have a branch this fixes feature "x" make sure that, that is all it does. Don't do a pullrequest that does anything else than what it says on the label. If there is a bunch of redundant changes in the branch, it will make it hard to track those changes later on since it's hidden under another pull request. So most likely your pullrequest will be rejected if this is the case.

Keep the amount of commits to a minimum
If you have a commit that fixes the issue, and then 5 commits to fix various mistakes / typos, you will be better off removing all the commits and then do a new commit, this will be much cleaner and easier to manage. You can look at the Troubleshooting section for how to remove a range of commits.

Write meaningful commit messages
No one can figure out what "newcommit" changes, and no one can track the change if an issue arises with it. Take your time to write a short but complete commit description such as:

> Improved performance in Player::update when the player stands on a chicken

Troubleshooting

This is general quality-of-life hacks that I have found very useful, I cannot guarantee that this is the 'official' way to do it, but for me it works.

Getting the id of a commit

You can use `git log` to find a list of all the commits ever made to the repository, you will see something like this:

```
commit 8ee943d9ba2a2856662fe1c7e5d6747794db2585
Merge: be32424 71d4082
Author: Lukas Joergensen <ljorgensen@winterleafentertainment.com>
Date:   Thu Sep 5 12:31:46 2013 -0700

    Merge pull request #1 from lukaspj/SideCamera

    Basic side camera

commit 71d4082810c4a26b8f4ee0cae44db18843d082d2
Author: Lukas Jørgensen <ljorgensen@winterleafentertainment.com>
Date:   Thu Sep 5 21:29:32 2013 +0200

    Fixed untracked files
```

It's that long string of numbers and characters that makes up the id of the commit.

Resetting your repository to an earlier state

If you have made a mistake and want to undo the changes made in a commit you will use the `git reset` command.
E.g.

```
git reset --soft 8ee943d9ba2a2856662fe1c7e5d6747794db2585
```

Notice how I used *--soft* here, there are 5 different modifiers, whereas I've only ever used 2, so to keep it simple I will only talk about those two:
--soft
This modifier resets the repository to an earlier stage, but does not undo changes to the files but instead marks them as "to-be-committed".

--hard
This modifier resets the repository to an earlier stage AND undoes all changes made to the files.

There is also some shortcuts to refer to earlier commits:

```
git reset --soft HEAD     // Resets the branch to current commit
git reset --soft HEAD~    // Resets the branch to the commit just before the current one
git reset --soft HEAD~2   // Resets the branch to the 3rd newest commit
git reset --soft HEAD~3   // Resets the branch to the 4th newest commit
// And so on..
```

So basically it goes:

```
HEAD->HEAD~->HEAD~2->HEAD~3->...->HEAD~n
```

(if you have a branch merged with the branch you are working on it gets a little more complicated than that, but I will assume you wont have a branch merged with your working branch)

Remove a range of commits

If you want to remove the commits themselves and not just revert the repository to an earlier stage (e.g. to make a clean PullRequest with just 1 commit), then you will use `git rebase -i`.
E.g.

```
git rebase -i HEAD~2
```

This will open up a text editor with options for what to do with each commit, if there is some commits you want to merge together, use the *"squash"* option to merge them together.
E.g.

```
pick 358412f New readme for v2.0
pick 4b644e4 Elastic ease fix
```

Change this to:

```
pick 358412f New readme for v2.0
squash 4b644e4 Elastic ease fix
```

To make the *Elastic ease fix* commit be merged into the previous commit, *New readme for v2.0*.