Git and GitHub Using Command Line

This document presents Git and GitHub using the command line.

The following video demonstrates the process.
Git and GitHub Using the Command Line, 32:15

Terminology

local - refers to the repository held in a directory on the computer on which you are developing.
remote - refers to the repository held on a git server (GitHub).
directory - a directory is the same as a folder.
folder - a folder is the same as a directory.

Branch types

master - the working version of the application
feature/<feature_name> - a capability implemented for an application
bugfix/<bug_descriptor> - a fix for a specified bug
hotfix/<hotfix_descriptor> - used to fix code in a released version

References

GitHub's git tutorial
Mastering Markdown - GitHub Guides
Git commit messages

Git/GitHub Process

Create local project directory

Create a directory on your development computer to hold the project.
The directory can be created by an IDE (such as Xcode or NetBeans) or it can be a directory you create at the command line, Finder (macOS) or File Explorer (Windows).

Git does not track empty directories. A directory must contain a file or files before it is included in the repository. If a project is created using an IDE then it will include files. If an empty directory is created as the starting point for the project, then at least one file must be created in the directory before a commit can occur.

If you create an empty directory, there are a couple of options for what file(s) can be placed in the directory at the start. Here are some options:

- Create an empty .gitkeep file. You can create an empty file using *touch .gitkeep* on Unix, Linux or macOS. Or, you can use a text editor to create the file.
- Put some starting files for the project in the directory. Generally, applications have some starting code that is required for any application that is to be developed.

You should avoid developing features in the master branch. A feature branch (see below) should be used when developing part of an application. Any starting files put initially in the project directory should be a rather generic starting point for the application.

Create repository on GitHub

Typically the name of the repository on GitHub is the same as the project directory. Spaces in names are converted into dashes when a repository is created.

When creating the repository on GitHub, add a repository name, description, README and a .gitignore. You can also create a license as well.

.gitignore

The .gitignore file specifies patterns for files and directories that git should ignore (not track).

gitignore documentation
.gitignore files

The .gitignore file should be edited to include a .DS_Store entry so that macOS .DS_Store files are not managed by git. For Xcode/Swift development choose the GitHib provided Swift .gitignore and edit it to add .DS_Store in the ##Other section of the file.

The .gitignore can be created initially in the GitHub repository or it can be created in, or added to, the local project directory. Don't initially create it in both the remote repository and the local project directory.
Put it in either one of those places initially.

Initialize git in local project directory to create local repository

```
git init
git add .
git commit -m "First commit"
```

Create a remote for the local repository

The *git remote add* below creates an alias for a remote repository on GitHub. The name *origin* is the alias for the GitHub remote repository clone HTTPS URL. The alias *origin* is the commonly used name for a project's remote repository.

```
git remote add origin <GitHub_remote_repository_clone_https_url>
```

The remotes for a repository can be listed using: *git remote -v*

Pull and rebase from origin

When the repository is created on GitHub it includes its own files and history. When git is initialized in the local project directory it also includes its own files and history. The following command pulls (retrieves) the files from the remote (origin) and merges the histories.

```
git pull --rebase origin master
```

Apply .gitignore to local and remote repositories

At this point the local repository has its original files and the files from the remote repository. These files may include files that are meant to be ignored according to the .gitignore file. The next step will remove the tracking of **all** files by git in the local project directory, add all the files to be tracked (ignoring any files specified in .gitignore), commit the files, and push them to the remote. When this is done both the local and remote repositories will track all files not in the .gitignore.

```
git rm -r --cached .
git add .
git commit -m "Applied .gitignore"
git push -u origin master
```

Create and checkout a branch to create a feature

When a developer adds a new ability/feature to an application they should create and checkout a feature branch off of master on which they will add the feature.

```
git branch feature/<feature_descriptor>
git checkout feature/<feature_descriptor>
```

The branches for a repository can be listed using: *git branch*
The current branch is the one with a * beside it.

Perform development for the feature

The current branch should be the feature branch. At this point the code is written and tested for the feature.

During development commits/pushes should frequently be made to track the code changes and save the changes to the remote.

```
git add .
git commit -m "<message describing changes>"
git push -u origin feature/<feature_descriptor>
```

Merge the feature branch into the master branch

The master branch should always be the working version of the application. Once a feature is finished and tested it needs to be merged into the master branch.

```
git checkout master
git merge feature/<feature_descriptor>
git push -u origin master
```

Fix a bug

When a bug needs to be fixed, a bugfix branch off of master should be used to make the code changes. Create and checkout a bugfix branch, make the code changes, commit and push the changes, test the code, then merge the bugfix branch into the master branch.

Create and checkout a bugfix branch.

```
git branch bugfix/<bug_descriptor>
git checkout bugfix/<bug_descriptor>
```

Make the code changes to fix the bug and test the code. While making the changes make commits/pushes frequently to track the code changes and save the changes to the remote.

```
git add .
git commit -m "<message describing changes>"
git push -u origin bugfix/<bug_descriptor>
```

Merge the bugfix branch into the master branch.

```
git checkout master
git merge bugfix/<bug_descriptor>
git push -u origin master
```