Git Cheat Sheet

Configuration

User Information

Git uses a set of user name and user email to sign the commit messages developed by the developer, as such these are required fields that are better configured globally when working from a personal or dedicated computer.

```
$ git config --global user.name "[your_name]"
$ git config --global user.email [your email]
```

Important to notice that if using your full name when setting the user name the command should not forget to use the "" to enclose a string that accepts spaces on it.

Text Editor

You can configure the default text editor that will be used when Git needs you to type in a message. If not configured, Git uses your systems default editor.

```
$ git config --global core.editor [editor]
```

Merge Tool

The right merging tool can save a lot of time and errors when having to merge non fast-forward codes. There is a plethora of merge tools available, however I strongly recommend meld for its intuitive use and simple but effective graphical capability.

```
$ git config --global merge.tool [tool_name]
```

Working with Remotes

Local and Remote Branches

Additional Tools

stashing

Stashing lets to record the current state of the working directory and the index locally without affecting the HEAD pointer. This is very useful when you want to save the code (you do not consider it ready to commit) but want to go back to a clean working directory. The command saves your local modifications away and reverts the working directory to match the HEAD commit.

```
$ git stash list
```

```
$ git stash apply stash@{[stash_number]}
```

logs

The command display a list of the commits in the repository. By default it shows the SHA-1 number , the commit message and the author of the commit. Of relevant use is the option `--oneline` which display a brief version of the above message description that consist of the SHA-1 number and the commit message

```
$ git log <options>
```

The command is ideal when trying to go back to previous states as its shows the SHA-1 number required for the checkout action.

remotes

```
$ git remote show [remote_name]
```

Show additional information about the remote and its relationship between remote and local branches. This command is ideal to list what remotes branches are being tracked and the pushing order of the local repository

gitk

Even though is not officially part of the Git software, the gitk tool is an excellent visualizer of a repository history

```
$ gitk [<options>] [<revision range>] [--] [<path>..
```

Of particular interest is the combination of revision ranges which will basically plot the differences between the given revision and the path