# Getting started with Git

GIT has three main stages a file can occupy.

* Committed - safely stored
* Modified - when changes are made since it was last committed.

i.e. changes have been introduced

* Staged - marks the file for the next commit cycle

Files not tracked are simply those that were not present in the last commit cycle.

To include a new file to track, it will be staged and committed as part of the next cycle.

Three stages:

.git directory (repository)

Working directory (checkout)

Staging Area (Index) - sits between working directory and repository, holding all changes wished to be committed.

# Installing Git

For windows, https://git-scm.com/download/win

For Mac, https://git-scm.com/download/mac

or homebrew - brew install git

**Verify the version using**

get --version

**Next, set the username using 'git config'**

git config --global user.name "Tony Breski"

git config --global user.email "tony.breski@gmail.com"

By omitting **--global** allows for local changes.

**git config --list** to show all settings

Use **git config user.name** to show specific values for a property

To get help, use **man git** or **git help <<topic>>**. Using **git help** will return a list of porcelain commands (most common). Plumbing command (scripting options) can be found separately using more detailed help.

# Creating a new REPO

Navigate to the project folder in a terminal/command window.

Use **git init** to create a new repository. This creates a new subfolder (.git) that is used for controlling the project.

# Sending code to a code host

GitHub, GitLab and BitBucket are common online code hosts.

Register an account on GitHub. You can create public or private repositories.

Add files to staging area:

**git add .**

Adds all files in the current folder into the staging area

To perform a commit:

**git commit -m "First commit"**

Git will show what updates/inserts took place.

Connect the repositories on the ones connected in Web portal & push our local code to cloud

**git remote add origin https:.....** (link to git repositories)

**git push -u origin master**

# Common GIT commands (Porcelain commands)

Check the status using:

**git status**

**File status**

Files can be tracked or untracked.

Tracked files can be committed, modified or staged.

Any committed files that get changed are moved to a modified state.

When we are happy with the changes, modified files can be staged for commit by adding them to the staged area. Once all changes are ready, the staged area can be committed.

To edit a file, use **open -a atom filename.txt** to use Atom as the default editor.

Use **git add <<filename>>** to include untracked files into the project.

Use **git reset HEAD <<filename>>** to remove it from the staging area.

**git diff --staged** shows differences between last commit and current staged area.

Add a **--no-renames** option to ignore suspected duplicates

Using **git diff** simply compares the working directory (modified).

# Committing changes

Using **git commit -a** allows you to bypass adding modified files to staging area and does it automatically. The **-a** automatically stages tracked changes.

Using **git commit -a -m "commit message"** allows to add a message on the commit.

# Extended Commands

**git push <<project>> <<branch>>** allows you to push local changes to cloud.

**git log** shows a reverse list of commits made against the project from cloud. Shows comments only.

Has lots of options that can be used with this command.

Some common ones include:

git log -20 where 20 is number of comments

git log –oneline provides a summary of comments

git log --state provides details of each commit

git log --patch gives a full breakdown, similar for git diff.

Commit messages often follow guidelines so that commit messages are meaningful.

See <http://chris.beams.io/posts/git-commit> for details.

# Removing files

**git rm <<filename>>** this tells git to remove the file. Git will stage it for removal. It is removed next time a commit is done. The file is also removed from the working area.

**git rm --cached <<filename>>** removes the file from working area and marks the file as untracked. The file remains in the working folder.

**git mv <<filename>> <<new-filename>>** is used to rename files.

# Branches

See <http://git-school.github.io/visualizing-git> for visualisation. This allows you to play around making commits/branches/etc and shows how the branch structure is created.

**git checkout -b <<branchname>>** creates a new branch from parent branch and checks out the code. HEAD points to the new branch created.

When creating new files from a master branch, the pending changes also remain.

**git stash** allows to save a copy of the working directory, useful for when switching between branches

Use **git stash list** to see the list of saved working areas.

Use **git stash show** to see what is in these

Use **git stash pop** to bring it back into the working directory

**git merge <<branchname>>** merges the <<branchname>> into the current branch (usually master into develop).

# Resetting Changes

Once changes are made in local it is possible to revert back from history to throw away changes. Be careful when using this.

**git reset --soft <id>** moves recent commits back to staging area. <ID> is the commit ID

**git reset --mixed <id>** default option and move changes back to working directory

(--mixed optional as it is the default)

**git reset --hard <id>** moves changes to trash and reverts everything back to last commit

# Pull from remote origin

**git clone <<url of repo>>** pulls down the repo into a folder of the repo name.

# Checkout older version of branch

**git checkout <<SHA-1 code>>** allows you to roll back code to a previous commit state. Effectively allows you to switch between commits.

# Merging code from Master into Branches

**git merge master** - pulls commits from Master into your current branch