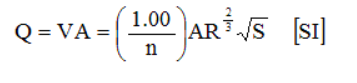
# Manning

Manning's Equation

One the most commonly used equations governing Open Channel Flow is known as the Mannings’s Equation. It was introduced by the Irish Engineer Robert Manning in 1889 as an alternative to the Chezy Equation. The Mannings equation is an empirical equation that applies to uniform flow in open channels and is a function of the channel velocity, flow area and channel slope.

Click here to view an interactive demo of Manning's Equation

Manning’s Equation:



Where:

Q = Flow Rate, (ft3/s)

v = Velocity, (ft/s)

A = Flow Area, (ft2)

n = Manning’s Roughness Coefficient

R = Hydraulic Radius, (ft)

S = Channel Slope, (ft/ft)

Under the assumption of uniform flow conditions the bottom slope is the same as the slope of the energy grade line and the water surface slope. The Manning’s n is a coefficient which represents the roughness or friction applied to the flow by the channel. Manning’s n-values are often selected from tables, but can be back calculated from field measurements. In many flow conditions the selection of a Manning’s roughness coefficient can greatly affect computational results.

# Creating 3d graphics

Using Unity Engine:

<https://unity.com/>

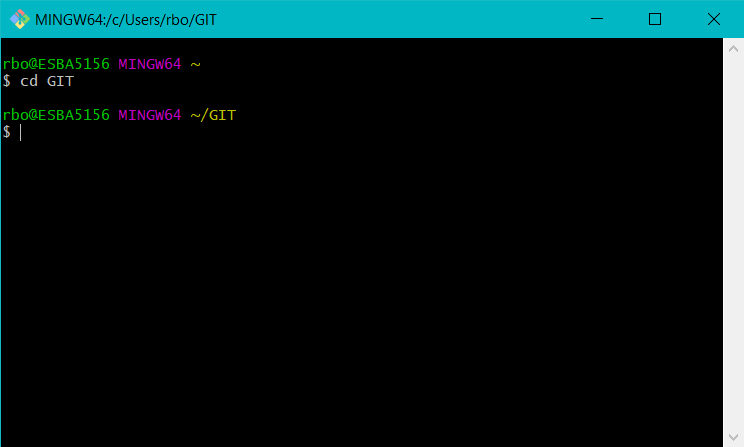
Using WPF:

<https://docs.microsoft.com/en-us/dotnet/framework/wpf/graphics-multimedia/3-d-graphics-overview>

# Creating a repository

Create the GIT repository making sure you are in the right folder.

La carpeta .git que se genera es la más importante ya que es la que tiene los archivos que mantienen la conexión con el repositorio remoto, las ramas y los archivos que pertenecen a repositorio tanto local como remoto. Al mismo nivel que la carpeta .git se deben encontrar las carpetas y código así como el archivo –gitignore.



Una vez en la carpeta deseada. Añadir el repositorio a GitHub es sencillo.

*git init* (git init <folder\_name> crea un directorio donde registra la carpeta .git)

*git add README.md* (añade todos los archivos del programa para realitzar el commit).

*git commit -m "first commit"*

git remote add origin <https://github.com/reb0rt081/Science-Engineering.git> (solamente es necesario ejecutar este comando una vez para crear la conexión “origin”).

git push -u origin master (mete el código local en el repositorio).Origin es el Identificador simbolico asociado al URL del repositorio remoto del que se ha clonado el repositorio local.

## git init [<git\_folder>]

Este comando genera un repositorio local GIT dentro la carpeta opcional (git\_folder) o en el directorio activo cuando se ejecuta el comando.

# Operaciones

## git add (.|<files>)

Añade o bien todos los archivos o aquellos archivos especificados listos para hacer commit.

Este comando se puede deshacer con **git reset [<files>]**. Es equivalente a “included changes” en TFS.

## git status [-s]

Muestra archivos modificados y que están añadidos al índice listos para hacer commit. Es equivalente a ver cual es el estado de los archivos modificados que están en included/excluded changes. Los que están “staged” están en “included changes”. Los “modified” están en “excluded changes”.

## git reset [<files>]

Borra los archivos del índice que pasan de estado “staged” (cuando están en el índice) a simplemente “modified”. Los cambios se mantienen en el directorio de trabajo. Es equivalente a “excluded changes” in TFS.

## git checkout [<files> | .]

Elimina todos los ficheros o el fichero especificado del directorio de trabajo. Se pierden los cambios.

## git checkout [commit\_id]

Descarga el código en la rama activa tal y como estaba en cierto **commit\_id**.

## git log [--oneline]

Muestra la historia con fechas y autor que nos lleva en la rama activa hasta el commit actual indicado.

### Ejemplo 1

rbo@ESBA5156 MINGW64 ~/GIT (master)

$ git log --oneline --graph --all

\* 788a4df (HEAD -> master, origin/master) Code is in and document updated

\* 01e16ad Training documentation

\* 0f4817f Hello world!

rbo@ESBA5156 MINGW64 ~/GIT (master)

$ git log 788a4df

commit 788a4dffb714521f33277f09c01375368d0309a4 (HEAD -> master, origin/master)

Author: Ribes Roberto <rbo@tgw-group.com>

Date: Fri Jun 21 13:31:58 2019 +0200

Code is in and document updated

commit 01e16adc4420e938ab6782b1ecee6bc149d23910

Author: Ribes Roberto <rbo@tgw-group.com>

Date: Thu Jun 20 23:54:40 2019 +0200

Training documentation

commit 0f4817f8d2b2a9d1c493c8559ddfc1924142b337

Author: Ribes Roberto <rbo@tgw-group.com>

Date: Thu Jun 20 23:48:37 2019 +0200

Hello world!

### Ejemplo 2

rbo@ESBA5156 MINGW64 ~/GIT (master)

$ git log

commit 788a4dffb714521f33277f09c01375368d0309a4 (HEAD -> master, origin/master)

Author: Ribes Roberto <rbo@tgw-group.com>

Date: Fri Jun 21 13:31:58 2019 +0200

Code is in and document updated

commit 01e16adc4420e938ab6782b1ecee6bc149d23910

Author: Ribes Roberto <rbo@tgw-group.com>

Date: Thu Jun 20 23:54:40 2019 +0200

Training documentation

commit 0f4817f8d2b2a9d1c493c8559ddfc1924142b337

Author: Ribes Roberto <rbo@tgw-group.com>

Date: Thu Jun 20 23:48:37 2019 +0200

Hello world!

## git show [<commit>]

Muestra las diferencias de un <commit> (o del último commit) con el anterior en la rama activa.

## git diff

Muestra diferencias de lo que hay en el directorio local de trabajo entre included/excluded changes y lo que esta comprometido en el servidor (diferencias entre lo modificado y el ultimo commit).

# Trabajando con ramas

Ten en cuenta que las ramas son punteros a ciertos commits que son importantes, en verdad una rama es un commit que queremos guardar y mantener.

## git branch -v

Muestra las ramas existentes en el repositorio.

## git checkout -b <branch\_name> <commit\_id>

Crea una rama a partir del **commit\_id** y le pone de nombre **branch\_name**. Hace que “HEAD” apunte a la nueva rama de manera que ya estas listo para trabajar sobre ella. El “HEAD” o tu directorio local cambian de manera que solamente tienes una rama funcional en tu directorio.

## git checkout <branch\_name>

Con esto cambiamos de rama y vamos al ultimo commit de la rama seleccionada.

## git log --oneline (--all|--graph)

Con esto se ven todos los commits en todas las ramas y no solamente en las activas.

## git merge [branch\_name\_2]

Estando en la rama activa **branch\_name\_1** ejecutamos este comando para integrar en un commit todo lo que hay en la rama especificada (**branch\_name\_2**) en la rama activa (**branch\_name\_1**).

Si hay conflictos el merge se queda en pausa y se modifican los archivos que tienen conflictos. La rama activa cambia en un stado “MERGING”. Deshacer conflictos y continuar con **commit**.

Si se quiere un commit automatico después de resolver conflictos: **git merge [branch\_name\_2] --continue**.

Si se quiere abortar merge si hay conflictos: **git merge [branch\_name\_2] --abort**.

## Fast-forward

Cuando el grafo de commits indica que la integración se ha realizado ya de la rama “A” la rama “B”, git realiza la integración de la rama “B” a la rama “A” con Fast-Forward, reutilizando el commit ya generado en la integración del punto anterior y simplemente avanza el puntero de rama a dicho commit, sin crear uno nuevo (commit automatico).

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator (sine)

$ git checkout master

Switched to branch 'master'

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator (master)

$ git merge sine

Updating 32a4394..7e9e4f4

Fast-forward

calculator.html | 7 ++++++-

1 file changed, 6 insertions(+), 1 deletion(-)

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator (master)

## git push --all [https://github.com/reb0rt081/my\_calculator.git]

Sube el código de todas las ramas al repositorio remoto que está puesto como ejemplo.

## git push origin (:|--delete )<rama\_remota>

Borra la rama remota en el repositorio remoto.

## git fetch –q [https://github\_repository] <rama\_origen > <rama\_destino\_copia>

Copia una rama origen en una rama destino (si no existe se creará) de un repositorio local.

## git clone [https://github.com/<account>/<repository\_name>](https://github.com/%3caccount%3e/%3crepository_name%3e) <local\_repository\_name>

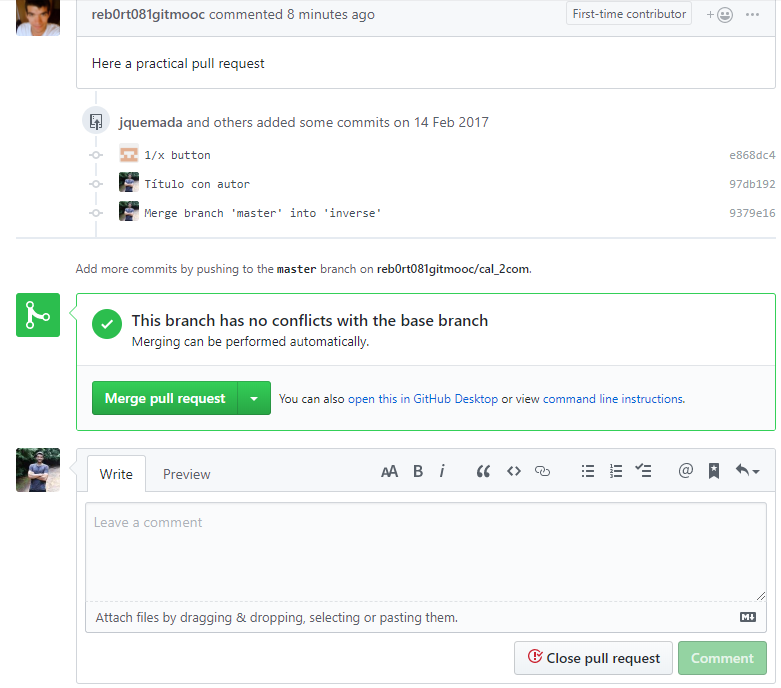
Una vez utilizado los comandos cd [folder\_name], cd .. y situado donde uno quiere clonar el repositorio solamente queda introducir el comando git clone seguido de la dirección del repositorio remoto y el nombre de la carpeta que se creará en la dirección local del ordenador (local\_repository\_name). Se descarga en local la rama master.

## Pull request

Desde nuestro repositorio clonado a través del botón “Fork” en **GitHub** podemos solicitar una integración de nuestros cambios en una rama desde una cuenta origen a la rama destino del repositorio original desde el que se hizo el “Fork” a nuestra cuenta.

## git pull –q (origin|<repositorio\_remoto>) pull/<id\_pull\_request>/head

Comando para integrar en la rama activa (la que sale en el comando) una “pull request” que se encuentra a la espera de ser integrada. Después se puede hacer el “git push”. <id\_pull\_request> es el numero asociado al pul request que estamos gestionando. Normalmente GitHub puede hacer el automerge desde la web:



Sin embargo, el comango “git pull –q origin pull/1/head nos ayuda a hacerlo manualmente”.

## git rebase <commit\_id>

El comando **git rebase** se utiliza para llevar todos los commits de una rama origen para llevarlos e integrarlos a un <commit\_id> especifico en una rama destino. Es necesario estar en la rama que queremos integrar. Si la rama origen es la misma que la rama destino entonces todos los commits se integrarán en uno solo con <commit\_id>. Huelga decir que los commits se integran de manera secuencial y que hay que solventarlos uno por uno.

Si queremos hacer un merge de una rama origen a una rama destino este comando puede ser utilizado.

En el caso practico de abajo vamos a coger el ultimo commit de la rama ops (e645235) y lo vamos a llevar al ultimo commit de la rama master. Primero hay que situarse en la rama master (git checkout master) luego llamar al procedimiento de rebase.

Git entonces iterará cambio por cambio para que hagamos integración de los cambios. Si hay cambios, hay conflictos hay que solventarlo y añadirlos y finalmente utilizat “git rebase --continue”, eso nos situará en el siguiente paso de la iteración.

Si no hay cambios con respecto a la interacción anterior e intentamos “git rebase -- continue” entonces GIT mostrará un error. Podemos utilizar “git rebase --skip” para pasar al siguiente paso o “git rebase --abort” para volver al principio.

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator\_2 (master)

$ git branch -v

\* master 9746274 Missing readme

ops e645235 1/x button

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator\_2 (master)

$ git rebase e645235

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator\_2 (master|REBASE 1/5)

$ git status

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator\_2 (master|REBASE 1/5)

$ git add calculator.html

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator\_2 (master|REBASE 1/5)

$ git rebase --continue

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator\_2 (master|REBASE 2/5)

$

En ocasiones haciendo rebase podemos modificar las líneas de comando que nos son dadas:

pick 1204dc8 Add Eva pending-tf

squash 71e69ce Add Eva tf

edit 9eaa103 Add Mary tf

# Rebase f6e660e..9eaa103 onto f6e660e (3 commands)

#

# Commands:

# p, pick <commit> = use commit

# r, reword <commit> = use commit, but edit the commit message

# e, edit <commit> = use commit, but stop for amending

# s, squash <commit> = use commit, but meld into previous commit

# f, fixup <commit> = like "squash", but discard this commit's log message

# x, exec <command> = run command (the rest of the line) using shell

# b, break = stop here (continue rebase later with 'git rebase --continue')

# d, drop <commit> = remove commit

# l, label <label> = label current HEAD with a name

# t, reset <label> = reset HEAD to a label

# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]

# . create a merge commit using the original merge commit's

# . message (or the oneline, if no original merge commit was

# . specified). Use -c <commit> to reword the commit message.

#

# These lines can be re-ordered; they are executed from top to bottom.

#

# If you remove a line here THAT COMMIT WILL BE LOST.

#

# However, if you remove everything, the rebase will be aborted.

#

# Note that empty commits are commented out

(Modifica el fichero con Notepad++ en la dirección que nos es dada en la ultima línea del output de la consola GIT Bash). Recuerda no editar lo que aparece en pantalla o tendras que volver a empezar desde git rebase --abort.

**:qa + <Enter\_Key>** cuando hayas cerrado el editor del archivo. Y continua.

El comando podría pedirte a continuación que renombraras el commit antes de continuar con **:qa + <Enter\_Key>** de nuevo.

# NPM

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator/mooc\_git-entrega1\_merge (master)

$ npm config get registry

<http://build-fxp-04.tgw.local:4873/>

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator/mooc\_git-entrega1\_merge (master)

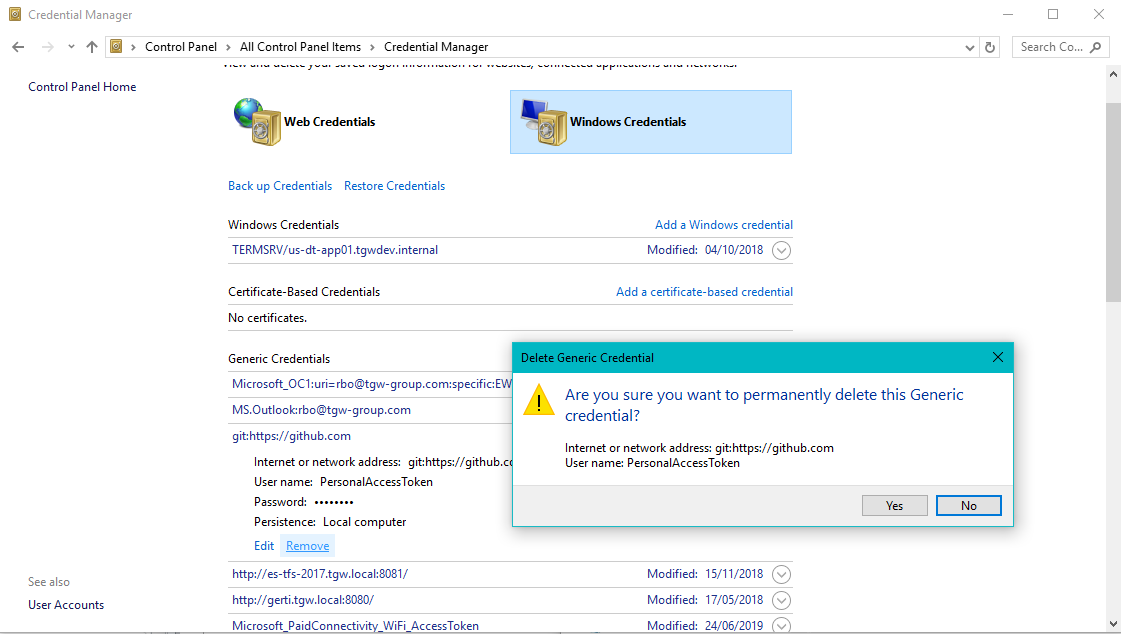
$ npm config set registry <https://registry.npmjs.org>

npm config set registry <http://build-fxp-04.tgw.local:4873/>

rbo@ESBA5156 MINGW64 ~/GIT/GIT\_Training/my\_calculator/mooc\_git-entrega1\_merge (master)

$ npm run checks reb0rt081

# Logging out/in git



Test: 1. Checking that the auxiliary repo https://github.com/reb0rt081gitmooc/cal\_2com exists

Score: 1/1

Remarks: https://github.com/reb0rt081gitmooc/cal\_2com found

Test: 2. Checking that the auxiliary repo contains the branch 'inverse'

Score: 1/1

Remarks: Found branch 'inverse'

Test: 3. Checking that the auxiliary repo branch 'inverse' includes 'inverse'

Score: 1/1

Remarks: Found 'inverse' in auxiliary repo branch 'inverse'

Test: 4. Checking that the auxiliary repo master branch includes 'inverse'

Score: 2/2

Remarks: Found 'inverse' in auxiliary repo master branch

Test: 5. Checking that the main repo https://github.com/reb0rt081/cal\_2com exists

Score: 1/1

Remarks: https://github.com/reb0rt081/cal\_2com found

Test: 6. Checking that the main repo master branch includes 'inverse'

Score: 0/2

Remarks: 'inverse' not found in the second to last commit from the main repo master branch.

Received: >>Título con autor<<

Test: 7. Checking that the main repo master branch includes 'pull request'

Score: 0/2

Remarks: 'pull request' not found in the last commit from the main repo master branch.

Received: >>Merge branch 'master' into 'inverse'<<

Final Result: 6/10