

# Reporte de laboratorio 0

Laura Rincón Riveros - B55863  
Esteban Vargas Vargas - B16998

23 de agosto de 2016

---

## Índice

<b>1. Introducción</b>	<b>2</b>
<b>2. Código</b>	<b>2</b>
2.1. C . . . . .	2
2.2. C++ . . . . .	3
2.3. Python . . . . .	4
<b>3. Makefile</b>	<b>5</b>
<b>4. GitHub</b>	<b>7</b>
<b>5. Conclusiones</b>	<b>14</b>

---

## 1. Introducción

En el presente laboratorio se implementó la función Suma en tres distintos lenguajes de programación. Conjuntamente se elaboró un *Makefile* para la compilación automática de los programas y Git para el control de versiones.

## 2. Código

### 2.1. C

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  /**
5   * @brief Suma de numeros recibidos por consola
6   *
7   *
8   */
9
10 int suma(int argc, char** argv){
11     double suma =0;
12     int i=0;
13     for (i=0; i<argc; i++){
14         suma+=atof(argv[i]);
15     }
16     printf("%f\n", suma);
17     return 0;
18 }
19
20 /**
21 * @brief Función main que llama a la función suma
22 *
23 *
24 */
25
26 int main(int argc, char** argv){
27     suma(argc,argv);
28     return 0;
29 }
30
```

Figura 1: Programa Suma en C

En la figura 1 se presenta el código del programa suma elaborado en C, donde se pueden observar dos funciones:

- Función suma: Esta recibe los números como parámetros ingresados por consola (*argv*) y los suma mediante un ciclo *for*.
- Función main: Esta función simplemente llama a la función suma.

## 2.2. C++

```
1  #include <iostream>↵
2  #include <stdio.h>↵
3  #include <stdlib.h>↵
4  #include <string>↵
5  using namespace std;↵
6  ↵
7  /**↵
8   * @brief Suma de numeros recibidos por consola↵
9   *↵
10  *↵
11  */↵
12  int suma(int argc, char** argv){↵
13  ..double suma =0;↵
14  ..int i=0;↵
15  ..for (i=0; i<argc; i++){↵
16  ....suma+=atof(argv[i]);↵
17  ..}↵
18  ..cout <<suma<< endl;↵
19  ..return 0;↵
20  }↵
21  /**↵
22  * @brief Función main que llama a la función suma↵
23  *↵
24  *↵
25  */↵
26  int main(int argc, char** argv){↵
27  ..suma(argc,argv);↵
28  ..return 0;↵
29  }↵
30
```

suma.cpp 1:1

Figura 2: Programa Suma en C++

En la figura 2 se muestra el código del programa suma elaborado en C++, donde se pueden observar, al igual que en el programa en C, dos funciones:

- Función suma: Esta recibe los números como parámetros ingresados por consola (argv) y los suma mediante un ciclo *for*.
- Función main: Esta función simplemente llama a la función suma.

## 2.3. Python

```
1  #!/usr/bin/python
2
3  import sys
4  """biblioteca para habilitar funciones de la consola"""
5
6  def Suma():
7      """funcion suma que recibe elementos desde la consola y los suma"""
8      counter = 0
9      """variable counter para guardar la suma de los elementos"""
10     array_nums= sys.argv
11     """arreglo que contiene lo que recibe la consola"""
12     for elements in range(1,len(sys.argv)):
13         """ciclos va desde el segundo elemento para eliminar ./suma.py hasta n cantidad de numeros ingresados"""
14         counter = counter + float(array_nums[elements])
15         """el ciclo recorre el arreglo, suma los elementos y los guarda en counter"""
16     print counter
17     """imprime el resultado"""
18 Suma()
19 """para ejecutar la funcion automaticamente"""
20
```

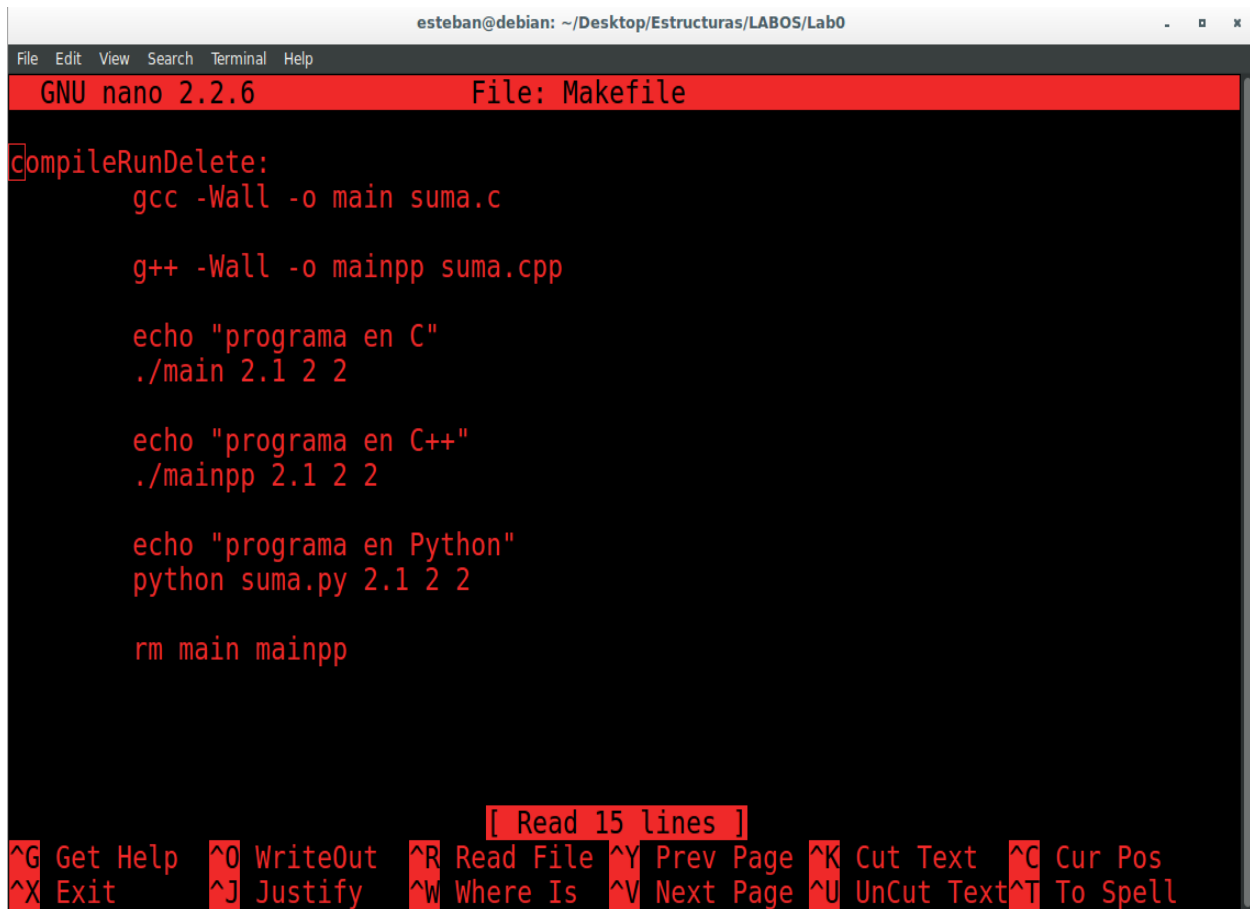
Figura 3: Programa Suma en Python

La figura 3 muestra el programa suma en Python. El cual, a partir de la lista de elementos `sys.argv` ingresados en la consola, filtra (mediante el ciclo) los números para sumarlos y acumular el resultado.

- Función suma: El cual, a partir de la lista de elementos `sys.argv` ingresados en la consola, filtra (mediante el ciclo) los números para sumarlos y acumular el resultado.

### 3. Makefile

Se realizó un archivo *Makefile* el cual, compila los archivos de código de C y C++, luego corre estos dos más el archivo de código de Python, y finalmente borra los ejecutables resultantes de la compilación de los códigos en los dos primeros lenguajes. A continuación, en la figura 4 se muestra el archivo.



```
esteban@debian: ~/Desktop/Estructuras/LABOS/Lab0
File Edit View Search Terminal Help
GNU nano 2.2.6 File: Makefile

compileRunDelete:
    gcc -Wall -o main suma.c

    g++ -Wall -o mainpp suma.cpp

    echo "programa en C"
    ./main 2.1 2 2

    echo "programa en C++"
    ./mainpp 2.1 2 2

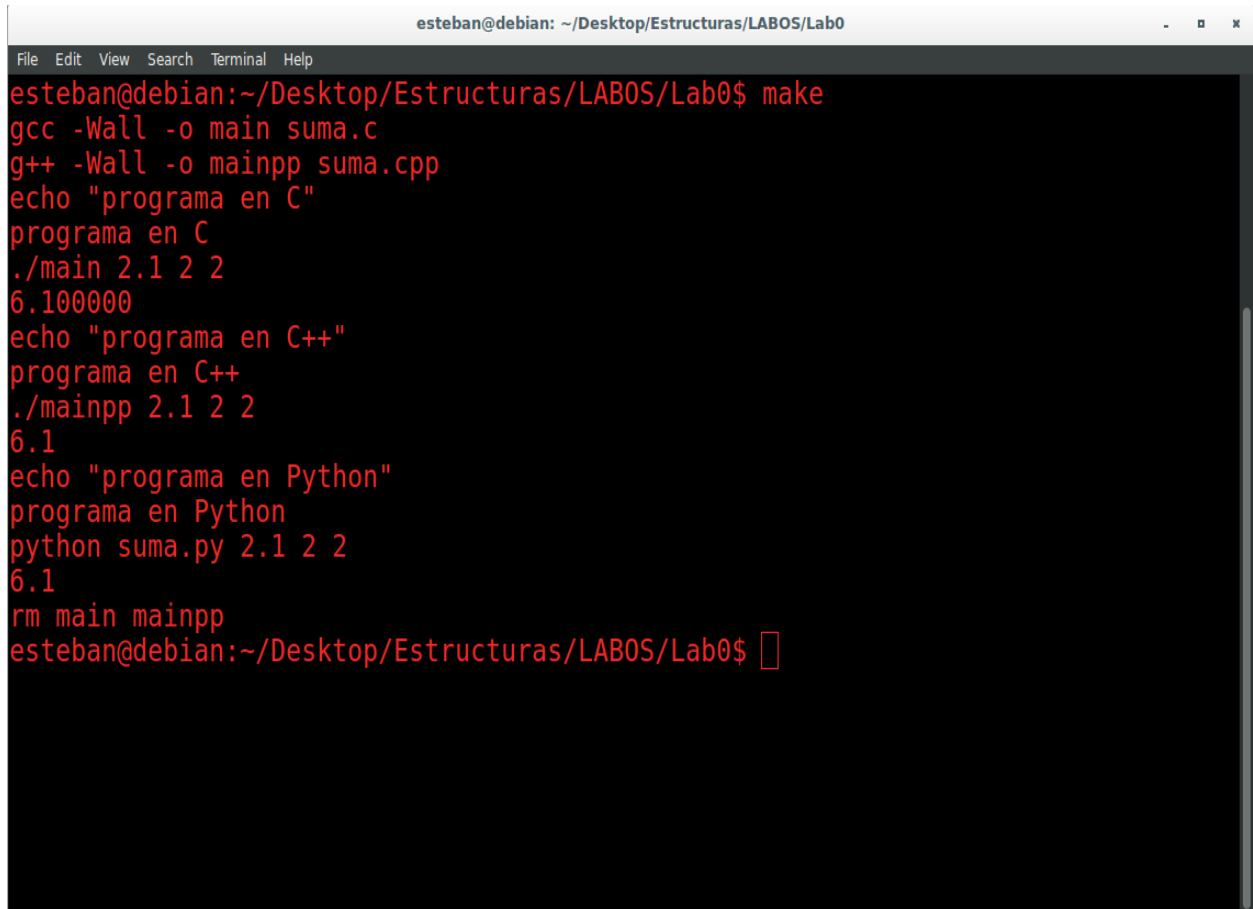
    echo "programa en Python"
    python suma.py 2.1 2 2

    rm main mainpp

[ Read 15 lines ]
^G Get Help ^O WriteOut ^R Read File ^Y Prev Page ^K Cut Text ^C Cur Pos
^X Exit ^J Justify ^W Where Is ^V Next Page ^U UnCut Text ^T To Spell
```

Figura 4: Makefile

Asimismo, en la figura 5 se ilustra el resultado del *Makefile*. En ésta se observa como los 3 programas entregan el mismo resultado para los mismos valores de entrada.



```
esteban@debian: ~/Desktop/Estructuras/LABOS/Lab0
File Edit View Search Terminal Help
esteban@debian:~/Desktop/Estructuras/LABOS/Lab0$ make
gcc -Wall -o main suma.c
g++ -Wall -o mainpp suma.cpp
echo "programa en C"
programa en C
./main 2.1 2 2
6.100000
echo "programa en C++"
programa en C++
./mainpp 2.1 2 2
6.1
echo "programa en Python"
programa en Python
python suma.py 2.1 2 2
6.1
rm main mainpp
esteban@debian:~/Desktop/Estructuras/LABOS/Lab0$
```

Figura 5: Resultado Makefile

## 4. GitHub

Se siguió el tutorial <https://try.github.io/levels/1/challenges/1>. A continuación se presentan imágenes que con la explicación de los principales instructivos.

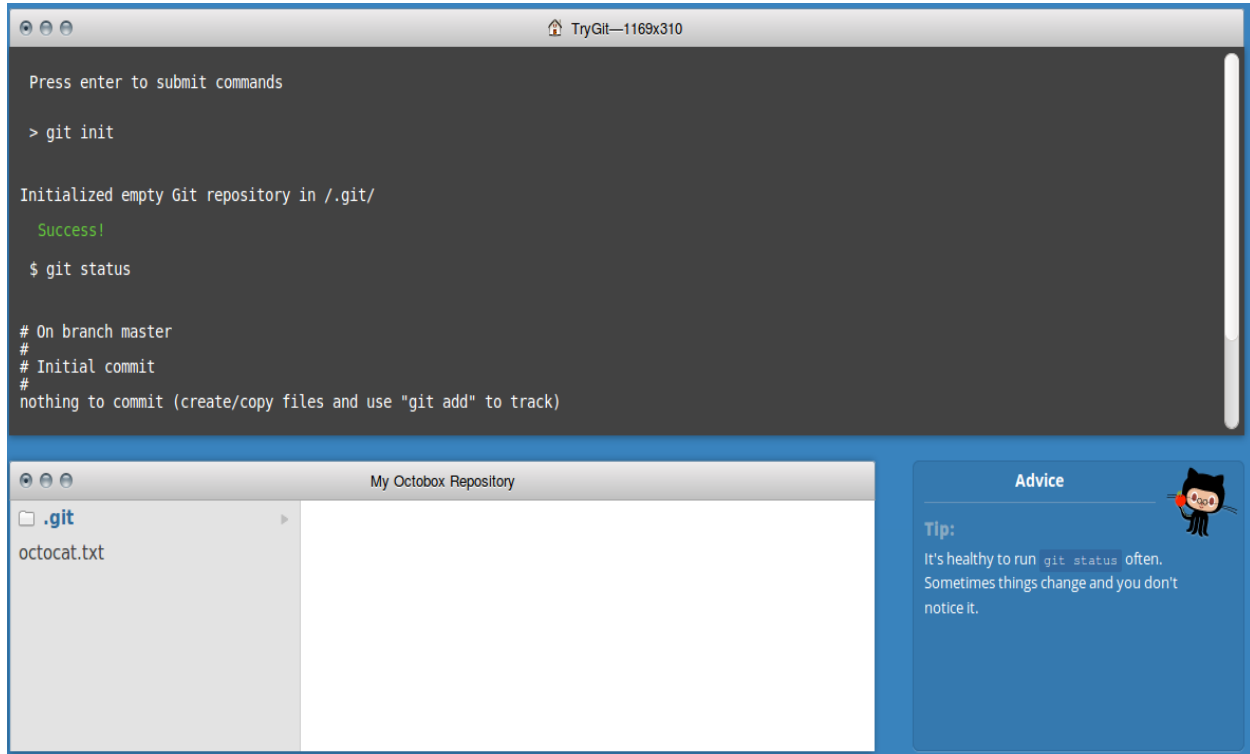


Figura 6: Seguimiento del tutorial de Git

La figura 6 muestra los comandos:

**git init:** para crear un nuevo repositorio.

**git status:** describe el estado del repositorio.

En la parte inferior, se puede ver el nuevo repositorio .git y los archivos en el repositorio.

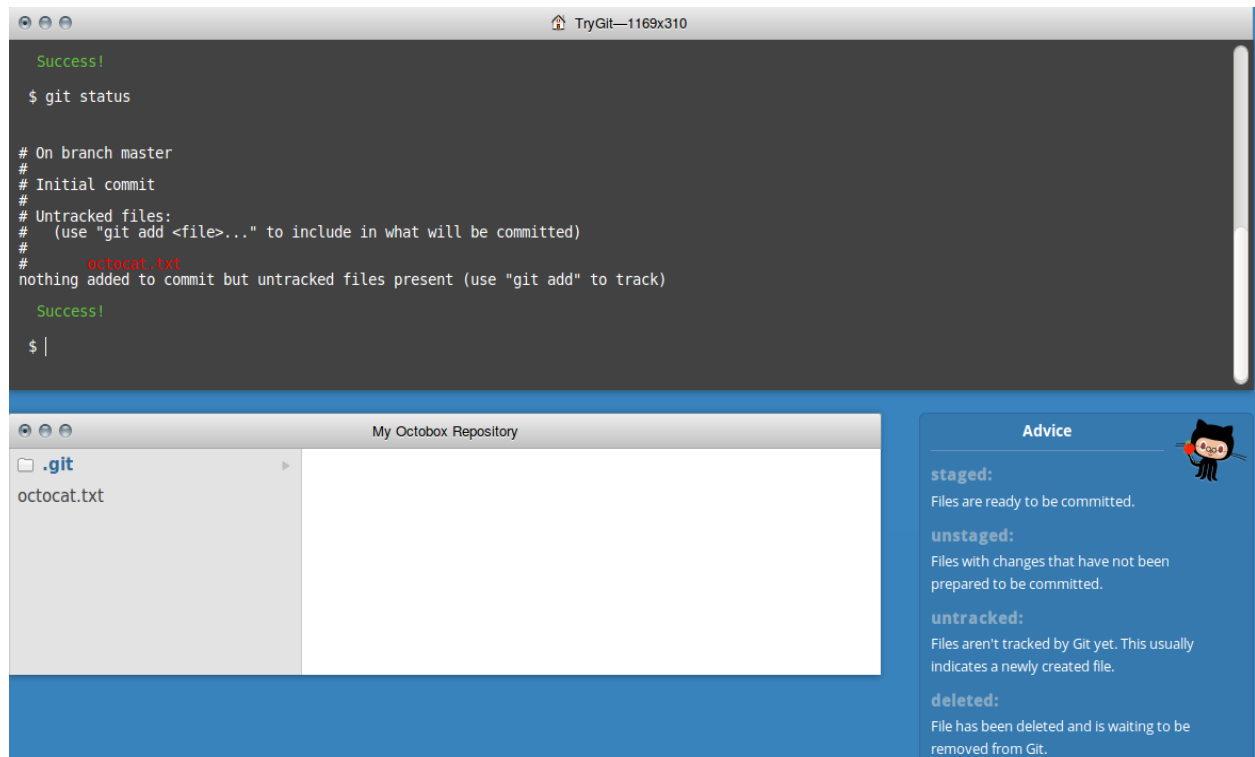
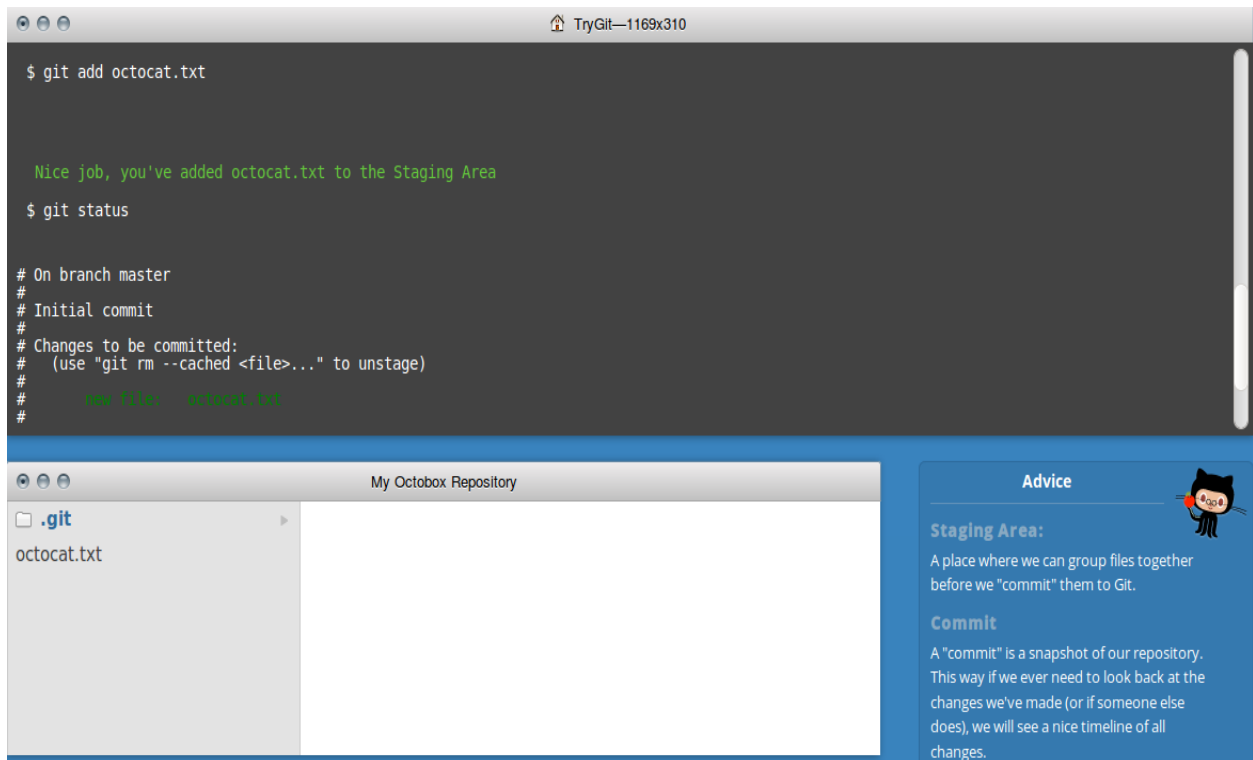


Figura 7: Seguimiento del tutorial de Git

La figura 7, se refleja el estado de `octocat.txt`, se necesita el comando **git add** para seguir los cambios del archivo.



Al aplicar **git commit -m "message"** el archivo está incluido en HEAD, el cuál, apunta al último commit realizado. El commit crea un panorama de los cambios realizados al repositorio, como una línea del tiempo. Como se muestra en la figura 8



The screenshot displays a Git tutorial interface. The top section is a terminal window titled "TryGit—1169x310" with a dark background. It shows the following commands and output:

```
$ git add octocat.txt

Nice job, you've added octocat.txt to the Staging Area

$ git status

# On branch master
#
# Initial commit
#
# Changes to be committed:
#   (use "git rm --cached <file>..." to unstage)
#
#       new file:   octocat.txt
#
```

Below the terminal is a file explorer window titled "My Octobox Repository". It shows a file tree with a folder named ".git" and a file named "octocat.txt".

On the right side of the interface is a blue sidebar titled "Advice". It contains two sections:

- Staging Area:** A place where we can group files together before we "commit" them to Git.
- Commit:** A "commit" is a snapshot of our repository. This way if we ever need to look back at the changes we've made (or if someone else does), we will see a nice timeline of all changes.

A small GitHub Octocat logo is visible in the top right corner of the sidebar.

Figura 8: Seguimiento del tutorial de Git

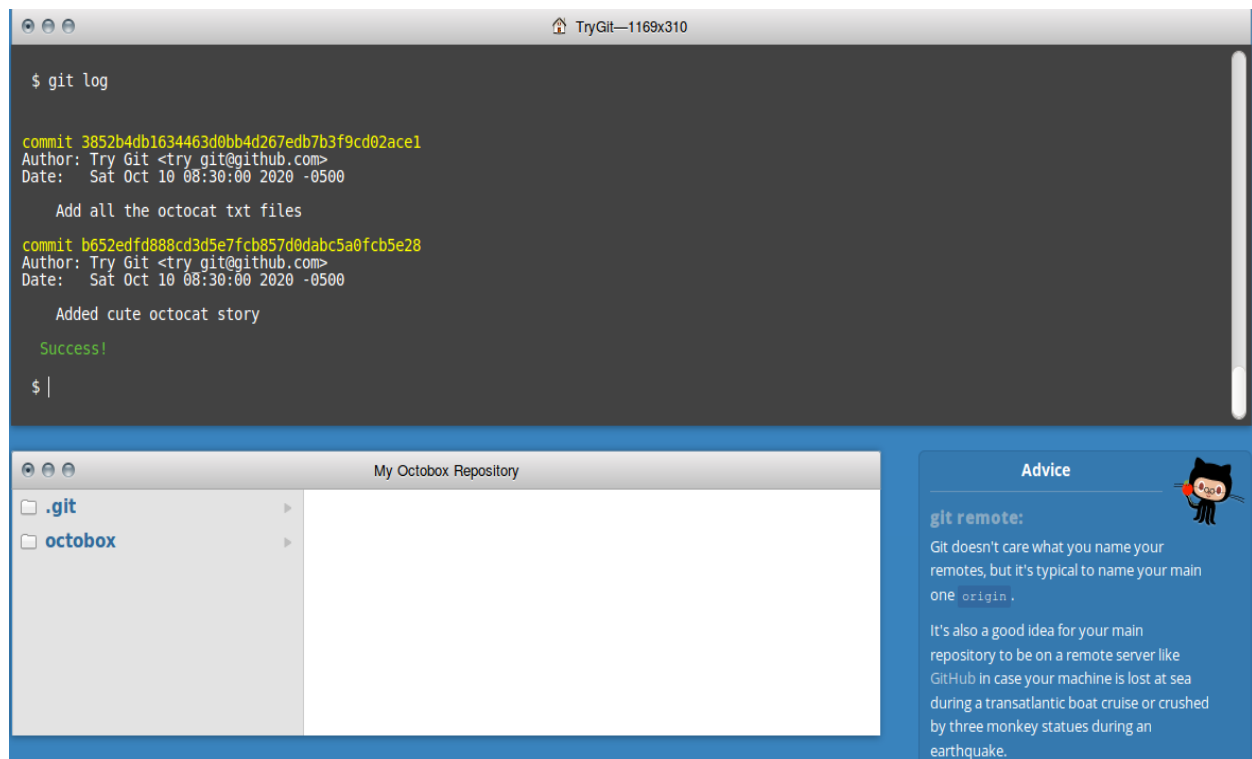


Figura 9: Seguimiento del tutorial de Git

La figura 9, muestra la ejecución del comando **git log** que imprime la etiqueta id, autor y fecha de los commits realizados.

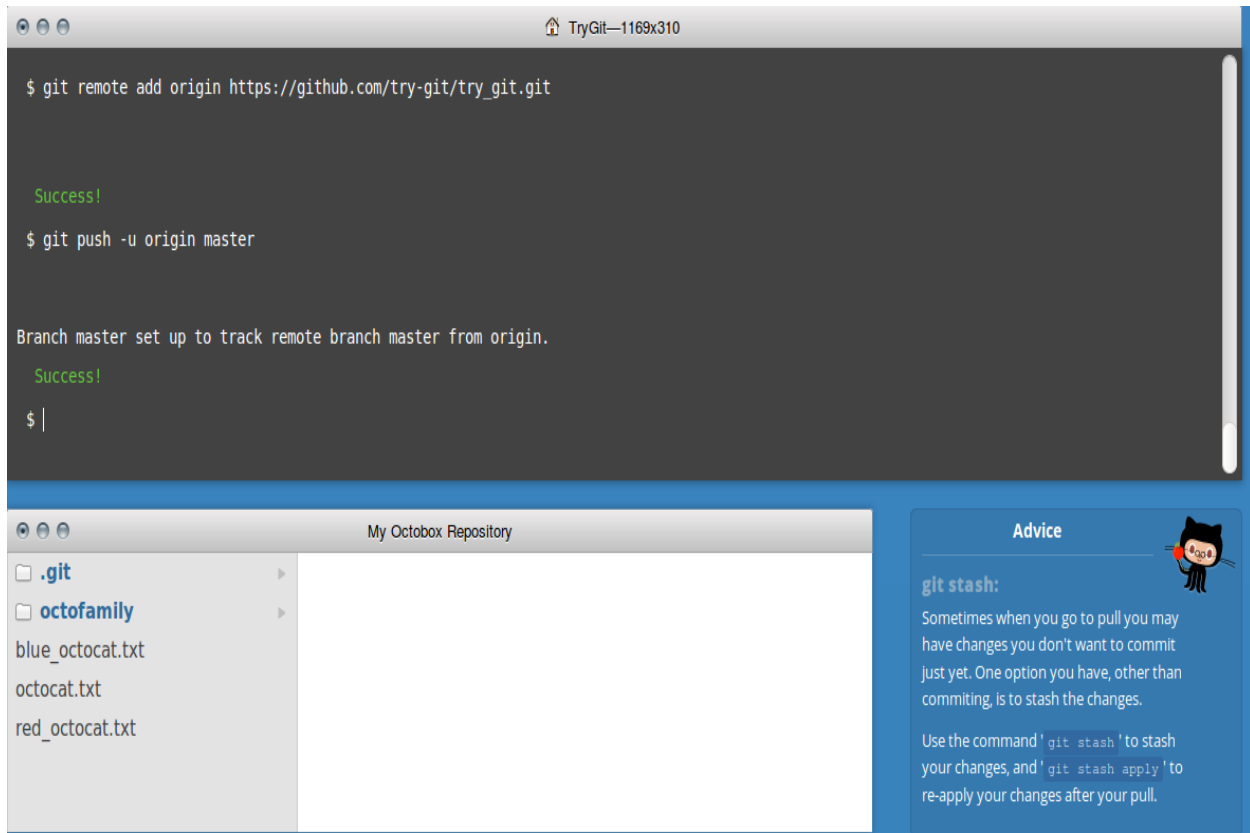


Figura 10: Seguimiento del tutorial de Git

A la rama **remote** se puede acceder desde cualquier servidor, por lo que es recomendable que el repositorio esté en el remote, como se ve en la figura 10.

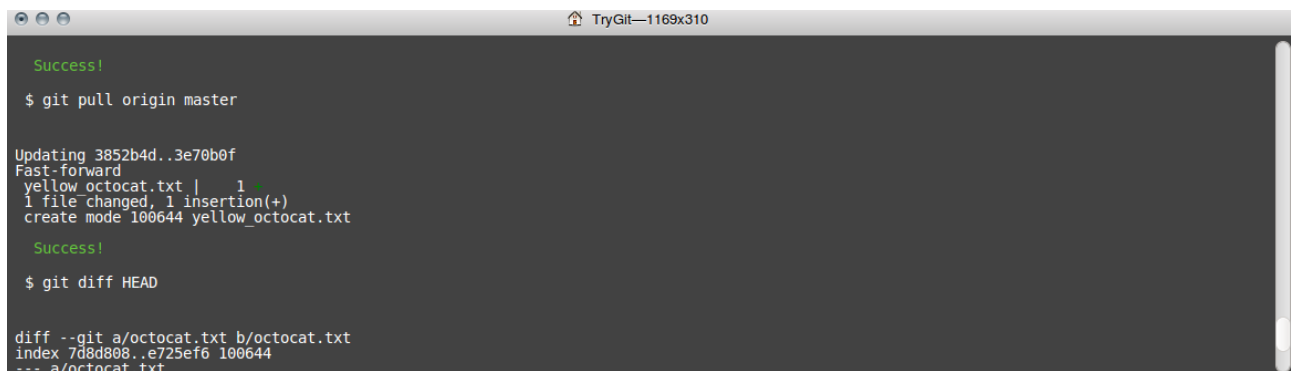


Figura 11: Seguimiento del tutorial de Git

La figura 11 muestra los comandos **git pull** y **git diff**, con los que se actualiza el repositorio local con el más reciente commit y se revisan los cambios, respectivamente.

En el caso de querer deshacer cambios, se requiere de los comandos **git reset** para cambiar la copia principal del repositorio y **git checkout** para remplazar el archivo por el anterior. Como se ilustra en la figura 12.

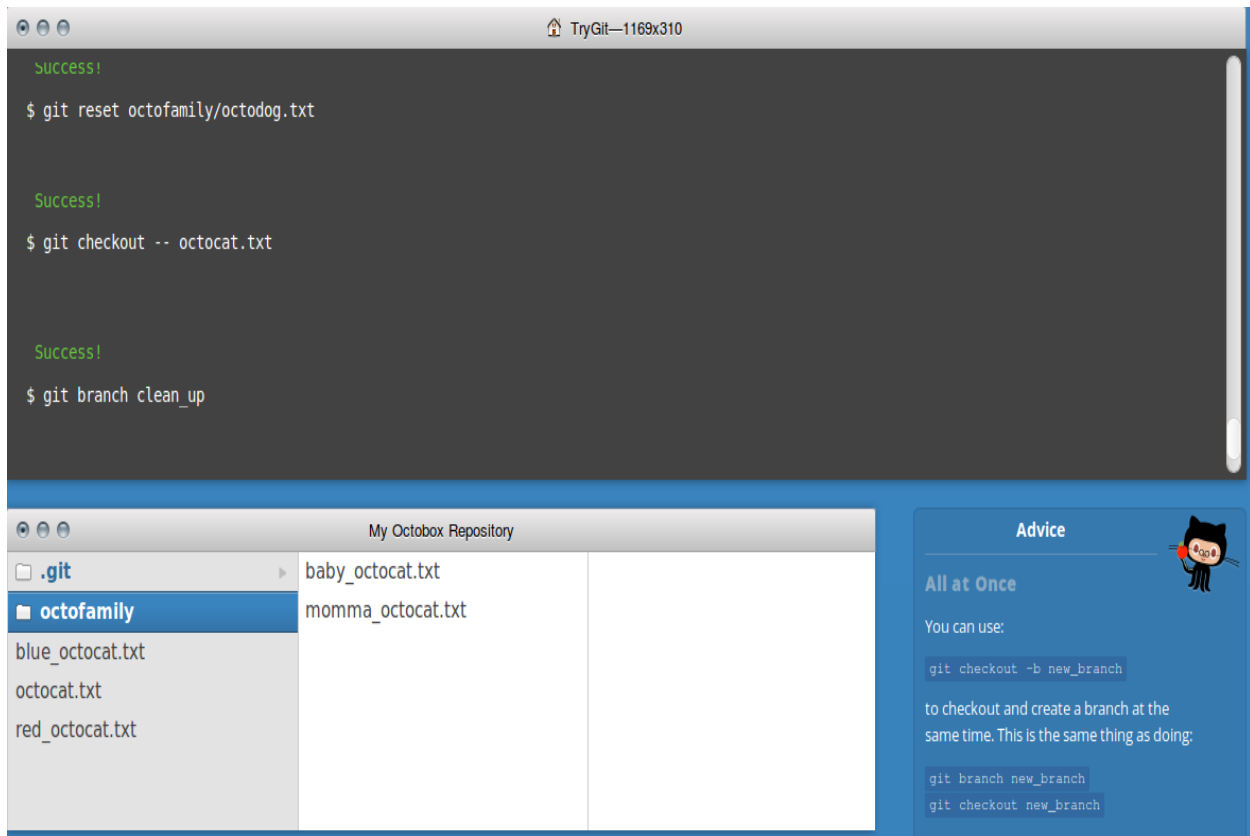


Figura 12: Seguimiento del tutorial de Git

Además, se muestra cómo se crea una nueva rama con **git branch**.

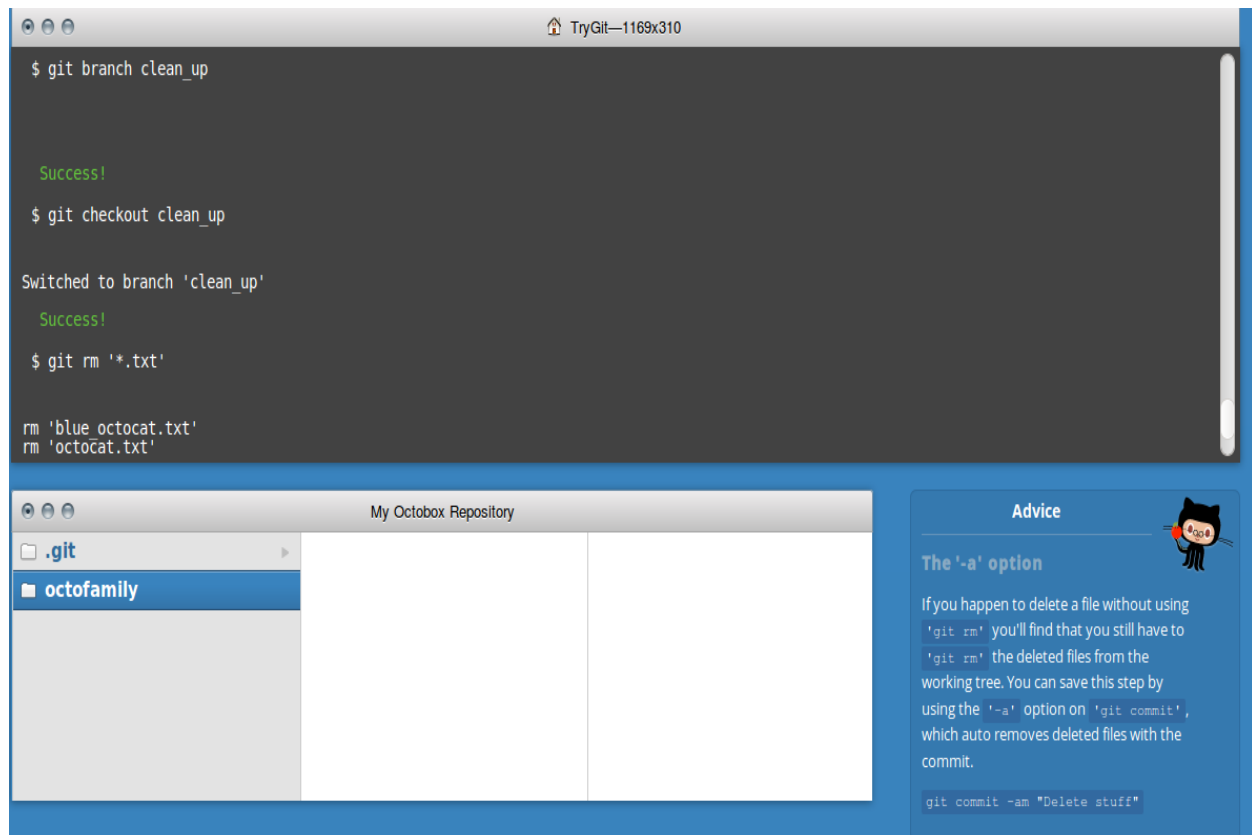


Figura 13: Seguimiento del tutorial de Git

En la figura 13, se emplea el comando **git rm** para eliminar los archivos del repositorio.

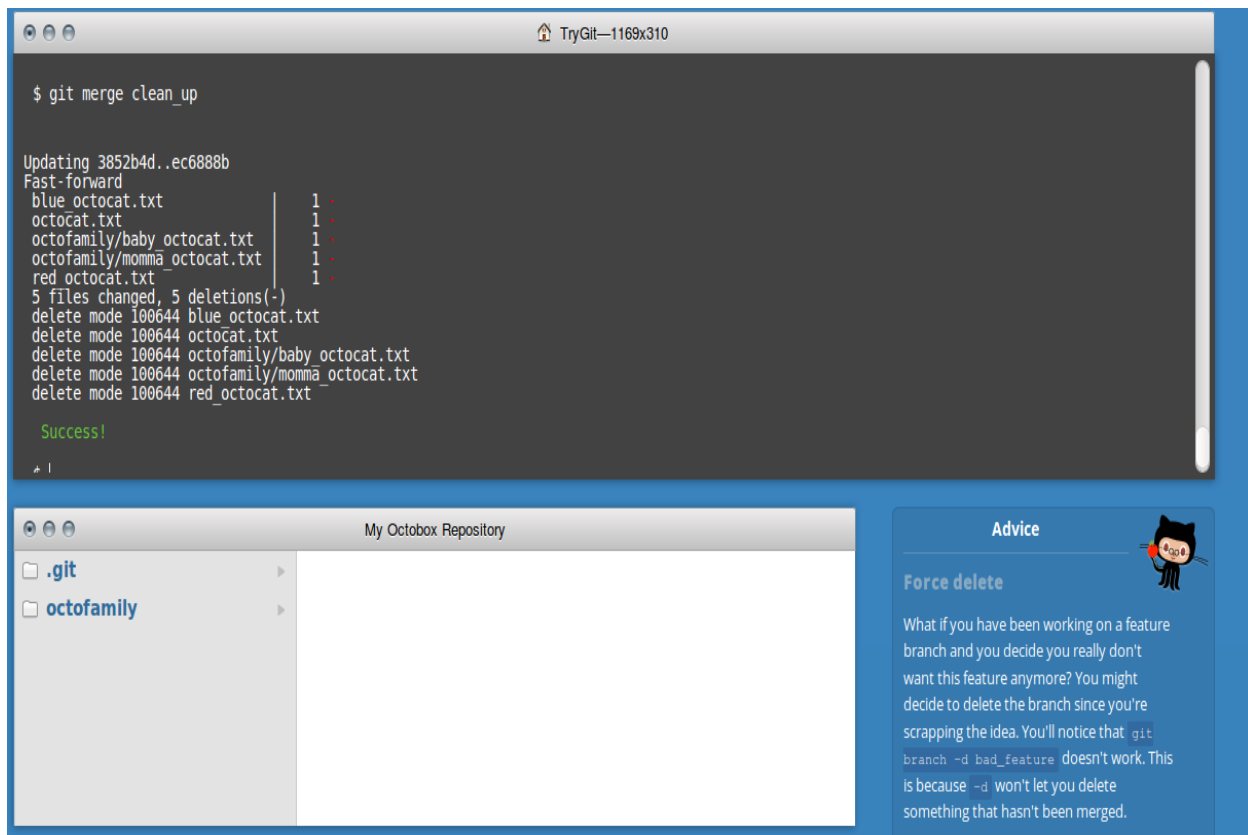


Figura 14: Seguimiento del tutorial de Git

Finalmente, la figura 14 muestra cómo se puede trabajar en una rama distinta a la principal y después fusionar las ramas.

## 5. Conclusiones

- Se implementó de manera exitosa una función suma en C, C++ y Python.
- Se logró automatizar la compilación y ejecución de los programas realizados, a través de un archivo *Makefile*.
- Gracias al tutorial proporcionado, se logró entender de manera elemental la herramienta de manejo de versiones *Git*.