|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  | |  |
|  | http://www.universidadescr.com/logos_dir/14_gr_uh_gr.png | | | | | |
|  |
|  |
|  |
|  |
|  |
|  |
|  |  |  |  | |  |
|  |  |  | Tarea I. | |  |
|  |  |  |  | |  |
|  |  |  | Curso: Programación II. | |  |
|  |  |  |  | |  |
|  |  |  | Profesor: | |  |
|  |  |  | Ing. Jorge Davián Aguilar Oviedo. | |  |
|  |  |  |  | |  |
|  |  |  | Alumna: | |  |
|  |  |  | María Fernanda González Sánchez. | |  |
|  |  |  |  | |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  | |  |
|  |  |  | 2016 | |  |

Diferencias entre C/C++/C#

|  |  |  |
| --- | --- | --- |
| **C** | **C++** | **C#** |
| Pedir un dato:  scanf("modificador", nombre de la variable);  Mostrar un dato:  printf("Dato: modificador", nombre de la variable); | Pedir un dato:  cin >> nombre de la variable;  Mostrar un dato:  cout << "Dato: " << nombre de la variable; | Pedir un dato:  Console.WriteLine(" ");  variable=Console.ReadLine();  Mostrar un dato:  Console.WriteLine(" " + variable); |
| La extensión .c se refiere a los archivos que se encuentran en lenguaje C. Restringe al compilador para que emplee solamente el lenguaje C y no permita instrucciones en lenguaje C++. | La extensión .cpp hace referencia a los archivos que se encuentran en lenguaje C++. El compilador se torna más flexible y permite que se usen instrucciones tanto en lenguaje C como en C++. | Para C# Visual Studio crea carpetas y archivos con distintas extensiones, identificando para que sirve cada una atreves del .cs, los cuales solo contienen código fuente en lenguaje C#. |
| C es un lenguaje de medio nivel ya que puede trabajar con el lenguaje máquina y también con el de alto nivel. | C++ es considerado de alto nivel, pero también tiene una considerable potencia para trabajar con lenguaje de bajo nivel. | C# pertenece a los lenguajes de alto nivel, que son más entendibles para la capacidad cognitiva humana. |
| En C, el programador es el encargado de liberar manualmente la memoria cuando no la utilice más. | En C++ es tarea del programador liberar estos espacios de memoria, y para ello se utilizan los destructores. | C# incluye un recolector de basura, lo que implica que el programador no necesita liberar memoria, esto pasaría a ser trabajo del GC. |
| C admite solo la programación estructurada. | C++ admite programación estructurada y la totalidad de programación orientada a objetos. | C# también admite la programación orientada a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. |

Git.

Es un software de control de versiones, el cual provee eficiencia y confiabilidad para el mantenimiento de versiones de aplicaciones cuando tienen un gran número de archivos de código fuente. Cada versión podría considerarse como una fotografía del estado de la colección en un momento determinado de tiempo, tanto la colección de archivos como su historia completa están guardados en un repositorio.

¿Cómo usar Git?

1. Configuración del nombre de usuario

Lo primero que se debe hacer es decirle a Git cual será nombre de la persona que hará uso de él:

$ git config --global user.name "Pedro Perez"

1. Configuración del email

Git hace uso del email del usuario cuando consolida la data, por esta razón necesitamos configurarlo:

$ git config --global user.email "pedroperez@ejemplo.com"

1. Podemos revisar que estén bien escritos haciendo uso del siguiente comando:

$ git config --list

user.name=Pedro Perez

user.email=pedroperez@ejemplo.com

1. Inicialización de un repositorio

Un repositorio no es más que una carpeta o conjunto de carpetas que contienen archivos.

Podemos crear una carpeta y luego iniciar el repositorio dentro, utilizando los siguientes comandos:

$ mkdir Ejemplo && cd Ejemplo

$ git init .

Con el comando git init se crea una carpeta oculta llamada .git y contiene toda la información necesaria para que podamos realizar las versiones de nuestro proyecto.

1. Agregar archivos al escenario

Con nuestro repositorio listo, queremos llevar el control de nuestro primer archivo.

$ touch Archivo1.txt // Creamos el archivo vacío

$ echo 'Hola Mundo' >> Archivo1.txt // Le agregamos texto al archivo

$ git add Archivo1.txt //colocamos el archivo en escenario

Al ejecutar el comando git add Archivo1.txt estamos confirmando que los cambios que realizamos sobre *Archivo1.txt* se quieren respaldar la próxima vez que consolidemos la data.

1. Consolidar la información.

Para consolidar el archivo previamente creado y puesto en escenario debemos utilizar el siguiente comando:

$ git commit -m "Commit Inicial"

Comandos básicos.

**a) Buscando Ayuda:**

1. git help *comando* ó git *comando* --help

    Muestra la ayuda para ese comando

**b) Creación de un repositorio:**

2. git init

    Crea un repositorio en el directorio actual

3. git clone *url*

    Clona un repositorio remoto dentro de un directorio

**c) Operaciones sobre Archivos:**

4. git add *path*

    Adiciona un archivo o un directorio de manera recursiva

      -f : Fuerza la eliminación de un archivo del repositorio

5. git mv *origen* *destino*

    Mueve el archivo o directorio a una nueva ruta

      -f : Sobre-escribe los archivos existentes en la ruta destino

**d) Trabajando sobre el código:**

6. git status

    Imprime un reporte del estado actual del árbol de trabajo local

7. git diff *[ruta]*

    Muestra la diferencia entre los cambios en el árbol de trabajo local

8. git add *path*

    Selecciona el archivo para que sea incluido en el próximo commit

9. git commit

    Realiza el commit de los archivos que han sido registrados (con git-add)

      -a : Automáticamente registra todos los archivos modificados

10. git clean

    Elimina archivos desconocidos del árbol de trabajo local

11. git blame *[archivo]*

    Muestra el archivo relacionado con las modificaciones realizadas

**f) Repositorios remotos:**

12. git fetch *[remote]*

    Trae los cambios desde un repositorio remoto

13. git pull *[remote]*

    Descarga y guarda los cambios realizados desde un repositorio remoto

14. git push *[remote]*

    Guarda los cambios en un repositorio remoto

15. git remote

    Lista los repositorios remotos

GitHub.

Github es un servicio de alojamiento de repositorios de software que ofrece diversas opciones para trabajar en equipo.

Como usar ¿GitHub?

**1. Crear un repositorio en GitHub**

Para crear un repositorio vamos al menú superior izquierdo y hacemos click en New repository.

El sitio GitHub pedirá que verifiques la cuenta desde el email de verificación que se envió a tu cuenta de correo. Después de ejecutar este paso volvemos al sitio web de GitHub donde ya podremos crear nuestro repositorio.

Al hacer click en el botón Create repository, inmediatamente nos va a llevar al repositorio. Como nuestro proyecto no tiene nada en su interior, no nos va a mostrar más que una ayuda de como subir archivos y proyectos.

**2. Administrar repositorio Git desde Windows**

Una vez creado el repositorio, el siguiente paso es subir archivos desde nuestra computadora personal, en la cual debemos instalar Git que nos permitirá conectarnos al repositorio de GitHub.

Se crea un directorio de trabajo en nuestra maquina local, en esta carpeta debemos inicializar los ficheros necesarios para que GIT funcione utilizando el comando **Git init**:

Una vez hecho esto, añadimos este fichero a GIT y hacemos nuestro primer commit.

Y ahora procedemos a decirle a nuestra copia local donde debe alojarse, en este caso en el de GitHub. El comando push envía estos cambios al servidor, en este caso indicando el repositorio donde va a alojarse.

[git push windows command line](http://www.app-softwarefactory.com/wp-content/uploads/2015/10/tutorial-github-windows-13.jpg)

Si miramos el repositorio desde la web de GitHub podremos observar los cambios que hicimos en el repositorio.

**4. Eliminar repositorio de GitHub**

Si queremos eliminar completamente un repositorio, simplemente abrimos nuestra cuenta en a la web de GitHub y vamos a Settings:

Se abrirá una nueva página y en la parte inferior, en Danger Zone encontraremos la opción Delete this repository.

Comandos Básicos.

* Checkout

Este comando sirve para traer los archivos del repositorio en el servidor a la máquina local.

En github:

git clone /path/to/repository

* Add

En github, puedes utilizarlo para empezar el seguimiento de archivos nuevos y para preparar archivos.

git add <archivo>

* Commit

Envía las modificaciones que se han hecho de forma local al servidor.

En github:

git commit -m “Commit message”

git push

En git nosotros necesitamos dos comandos para hacer el commit, el primero funciona para notificarle a git que archivos se modificaron y el mensaje que tendrá la nueva versión del proyecto y el segundo hace la actualización.

* Update

Actualiza las modificaciones que se han hecho en el servidor para que se vean reflejadas de forma local.

En github:

git pull

Al hacer el comando anterior git también ejecutara el comando: “git merge <branch>” esto para combinar los archivos, no siempre será posible y se podrán producir conflictos.