

WebDev - Ayudantía 02

Nicolás Chirino - Desarrollo Web y Móvil 2024-02

¿Qué es Git(hub)?

Git es un sistema de control de versiones, el cuál utilizaremos a lo largo de toda nuestra carrera como ingenieros informáticos. Este sistema nos permite ir guardando "checkpoints" a medida que vamos modificando nuestro código, así podemos rastrear los cambios y colaborar con otras personas en simultáneo.

Por otro lado, **Github** es una plataforma en la nube la cuál nos permite guardar nuestros proyectos de git, compartirlos y colaborar en otros proyectos de forma gratuita. No es la única plataforma que realiza esto, pero es la más popular a día de hoy y la que usaremos en este curso.

Instalar Git

- **En Windows:** [Descargar instalador](#).
- **En Linux:**

```
sudo apt-get update  
sudo apt-get install git
```

- **En Mac:**

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/  
brew install git
```

Configurar Github

En primer lugar, necesitarán crearse una cuenta en [Github](#), para esto les recomiendo que lo hagan con su correo personal y posteriormente agreguen su correo universitario como segundo email. ¿Por qué agregar el correo universitario? Porque de esta forma pueden postular al [Student Developer Pack](#), el cuál les otorgará Github Pro junto a muchos beneficios, entre ellos


Github Copilot Pro en VSCode, créditos gratis en plataformas de Cloud, cursos de programación, IDEs profesionales como JetBrains, entre otros.

Application

Required fields are marked with an asterisk *

What is the name of your school? *

Note: If your school is not listed, then enter the full school name and continue. You will be asked to provide further information about your school on the next page. **A minimum of two characters is required to find your school.**

 You have verified the email address **nicolas.chirino@mail.udp.cl** on your GitHub account. That academic domain is associated with the school **Diego Portales University**.

Select this school

We chose this school based on your email. If this isn't your school, please [add and verify another school-issued email address](#) in your account settings - or select another school.

When you click "Continue" you will be prompted to share your location with us. Providing your current location helps us verify your affiliation with your chosen school.

Continue

Luego, configuraremos nuestra cuenta para poder acceder mediante SSH, para esto seguiremos el siguiente [Tutorial de Github](#), el cuál provee los comandos para Windows, Mac y Linux. Nos fijaremos en el apartado de "[Generar nueva clave SSH](#)" y posteriormente en "[Agregar tu clave al ssh-agent](#)". En la terminal de Windows (Powershell), son los siguientes comandos:

```
ssh-keygen -t ed25519 -C "tu@mail.cl"

# Acá apretamos enter, enter, enter

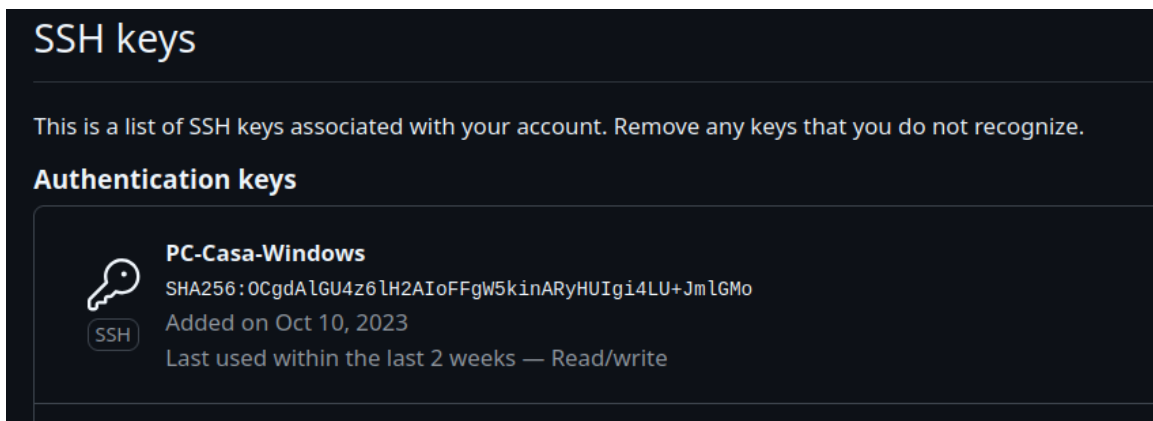
Get-Service -Name ssh-agent | Set-Service -StartupType Manual

Start-Service ssh-agent

ssh-add c:/Users/TU_NOMBRE/.ssh/id_ed25519
```

```
# Con esto ya está listo, ahora la copiamos con el comando de  
  
cat ~/.ssh/id_ed25519.pub | clip
```

Finalmente, agregaremos la clave SSH a nuestra cuenta de Github: Apretando nuestro ícono de perfil > Configuración > Llaves SSH y GPG > Nueva llave SSH. Esto está también detallado en [Otro Tutorial de Github](#). Si hicimos todo correctamente, se verá algo así:



Crear un repositorio

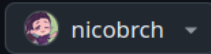
Un repositorio es el lugar en dónde guardaremos todo nuestro código. Para crear uno, apretamos el ícono de "+", unos botones al lado de nuestro ícono de perfil, y apretamos "Nuevo repositorio". Esto nos llevará a la siguiente interfaz, en dónde rellenaremos los campos:

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)

Required fields are marked with an asterisk (*).

Owner *



Repository name *

/ webdev

✔ webdev is available.

Great repository names are short and memorable. Need inspiration? How about **crispy-fortnight** ?

Description (optional)

Ejemplo de repositorio



Public

Anyone on the internet can see this repository. You choose who can commit.



Private

You choose who can see and commit to this repository.

Initialize this repository with:



Add a README file

This is where you can write a long description for your project. [Learn more about READMEs.](#)

Add .gitignore

.gitignore template: None ▾

Choose which files not to track from a list of templates. [Learn more about ignoring files.](#)

Choose a license

License: None ▾

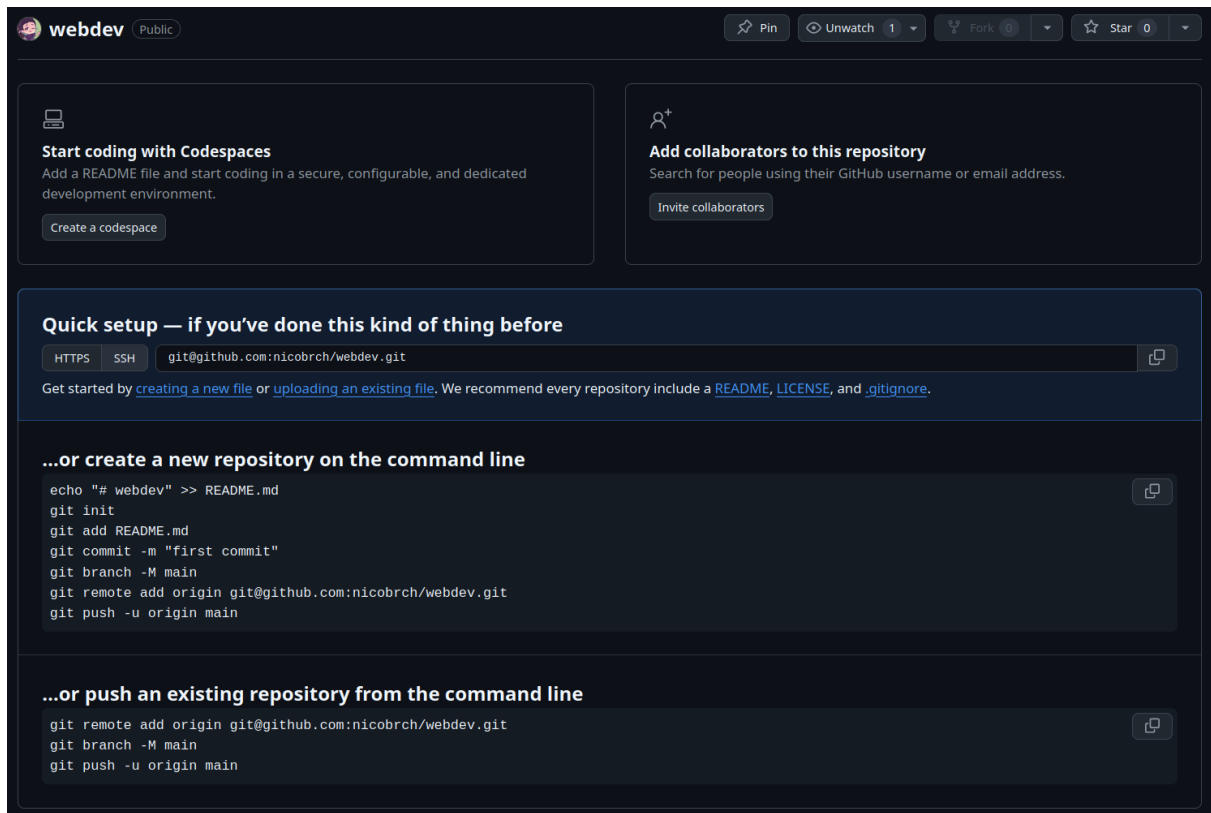
A license tells others what they can and can't do with your code. [Learn more about licenses.](#)



You are creating a public repository in your personal account.

Create repository

Una vez creado, nos aparecerá esta interfaz, la cuál nos indicará los comandos a utilizar



Y seguiremos los pasos de la sección “...or create a new repository on the command line”.

```
echo "# webdev" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin git@github.com:nicobrchr/webdev.git
git push -u origin main
```

Listo! ya está configurado nuestro repositorio y se ha subido correctamente el archivo “README.md”.

El “Gitflow”

Para utilizar git y no morir en el intento, seguiremos la metodología de Gitflow, que es una metodología que nos ayuda a utilizar git de una forma más ordenada, utilizando estándares y buenas prácticas.

comando `git commit -m "Mensaje"`. Intentar siempre poner mensajes descriptivos.

5. **Subir los cambios.** Para que todos podamos trabajar colaborativamente, será necesario que subamos nuestros cambios a la nube. ANTES DE SUBIR ALGO, VERIFICAR QUE ESTAMOS EN LA RAMA CORRECTA (feature). Finalmente, usaremos el comando `git push` para subir los cambios.
6. **Mergear los features.** Una vez que el desarrollo de un feature (en su respectiva rama) ya está listo, podemos cambiarnos a la rama develop, y ejecutar el comando `git merge <feature>` para incorporar el nuevo feature a su rama respectiva.

Y con esto ya podemos trabajar!

Repasando Javascript

Arreglos:

```
const frutas = ["Manzana", "Mango", "Pera"]
const sueldos = [100, 500, 250]
const personas = [
  {
    nombre: "Jose",
    edad: 22
  },
  {
    nombre: "Elisa",
    edad: 21
  }
]
```

Modificar elementos:

```
frutas[1] = "Piña"
// Error! dado que frutas es una variable CONST
let verduras = ["Lechuga", "Apio", "Brocoli"]
verduras[1] = "Alcachofa"
```

Buscar elementos:

```
// includes()
// Verificar si un valor existe en un array. Retorna V o F.
console.log(frutas.includes("Mango"))

// some()
// Verificar si existe valor dada una función. Retorna V o F.
const esPar = (valor) => valor % 2 == 0
console.log(sueldos.some(esPar))

// find()
// Encontrar un valor, si existe, dada una función. Retorna e
const sobre300 = (valor) => valor > 300
console.log(sueldos.find(sobre300))

// filter()
// Encontrar todos los valores, si existen, dada una función.
const masLargo = (valor) => valor.length > 4;
console.log(verduras.filter(masLargo))
```

ExpressJs, Javascript en el Backend

ExpressJs es un framework web para crear aplicaciones (especialmente API Rest) utilizando NodeJs, y es lo que usaremos a lo largo de este curso.

Definamos algunos conceptos:

- **API:** Application Program Interface, son mecanismos que permiten a dos componentes de software comunicarse entre sí mediante un conjunto de definiciones y protocolos.
- **API Rest:** Lo mismo que una API, pero diseñada para estar siempre funcionando y estar disponible para permitir la comunicación.
- **HTTP:** Protocolo de la capa de aplicación, el cuál utiliza el framework de ExpressJs para establecer la comunicación. Este posee los siguientes "verbos":
 - **GET:** Para obtener información. No causa efectos secundarios en el servidor.
 - **POST:** Para crear información. Debe crear un nuevo recurso.

- **PUT:** Para actualizar información. Cambia los valores de un recurso.
- **DELETE:** Para borrar información. Elimina un recurso.

Ok, entiendo los conceptos, pero ahora, ¿Cómo creo mi aplicación web?

Instalar Express usando NodeJs:

```
npm install express
```

Crear aplicación básica:

```
import express from "express"
const PORT = 3000

const app = express()

app.get("/", (req, res) => {
  res.send("Hola Mundo!")
})

app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`)
})
```

REQ y RES, ¿Qué son?:

```
// REQ
// Objeto que contiene información HTTP que recibe la app de
app.get("/cats", (req, res) => {
  console.log(req.url) //logs "/cats"
  console.log(req.method) // logs GET
  console.log(req.headers) // logs all headers of the request
})

// Usar parámetros en la petición REQ
app.get("/cats/:id", (req, res) => {
  console.log(req.params.id) // logs ID
})
```

```
// RES
// Objeto que con el que enviaremos respuestas mediante HTTP.
app.get("/cats/:id", (req, res) => {
  const cats = ["Sonic", "Chimuelo", "Serafina"]
  res.send({
    cats
  })
})
```

Práctica

Para practicar y saber que entendiste correctamente todo lo explicado en esta ayudantía, haz el siguiente ejercicio. Crea una aplicación usando Javascript, NodeJS y Express, la cuál contenga información de 5 Pokémon en una arreglo de objetos tipo const. Incluye valores clave para identificar cada Pokémon, como el nombre, tipo, nivel, entre otros. Crea un endpoint GET en la URL `/pokemon/:nombre` la cuál busque al Pokémon por su nombre, y devuelva la información como respuesta HTTP. Finalmente, sube los archivos a un repositorio de Github público.