**Instalación de Ruby, RoR en Mac**

**Introduccion**

Al igual que en la instalación del entorno para Linux, vamos a usar el framework de configuración oh-my-zsh aprovechando además que en Catalina, nuestra shell por defecto es zsh. Para hacerlo debes en la terminal de comandos colocar la siguiente instrucción, que encontrarás en la página oficial de github de oh-my-zsh <https://github.com/ohmyzsh/ohmyzsh#via-curl>

~ % sh -c "$(curl -fsSL https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"

A continuación te mostraré una respuesta completa de este comando (debes ver esto después de la ejecución del comando incluyendo la pregunta de cambio de consola)

\_\_ \_\_

\_\_\_\_ / /\_ \_\_\_\_ \_\_\_ \_\_ \_\_ \_\_\_\_ \_\_\_\_\_/ /\_

/ \_\_ \/ \_\_ \ / \_\_ `\_\_ \/ / / / /\_ / / \_\_\_/ \_\_ \

/ /\_/ / / / / / / / / / / /\_/ / / /\_(\_\_ ) / / /

\\_\_\_\_/\_/ /\_/ /\_/ /\_/ /\_/\\_\_, / /\_\_\_/\_\_\_\_/\_/ /\_/

/\_\_\_\_/ ....is now installed!

Please look over the ~/.bashrc file to select plugins, themes, and options.

p.s. Follow us on https://twitter.com/ohmyzsh

p.p.s. Get stickers, shirts, and coffee mugs at https://shop.planetargon.com/collections/oh-my-zsh

Al final de tu proceso deberás ver tu consola con un nuevo estilo como el siguiente:

➜ ~

Después de instalar oh-my-zsh debemos instalar un gestor de paquetes, en nuestro caso usaremos Homebrew (ampliamente usado en el mundo de desarrollo de software), puedes seguir las indicaciones de la instalación desde la siguiente pagina <https://brew.sh/> y digitar la siguiente instruccion:

➜ ~ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/**install**/**master**/**install**.sh)"

Una vez instalado homebrew usamos rbenv como gestor de versiones de Ruby, de igual manera que lo hicimos para la instalación del ambiente de Linux (puedes ir la sección Instalando RBENV en la lectura **Instalación Entorno de Desarrollo** para que obtengas más información).

➜ ~ **brew install** rbenv ruby-**build**

Luego debemos añadir al archivo .bashrc una configuración de inicialización del RBENV

➜ ~ echo 'if which rbenv > /dev/null; then eval "$(rbenv init -)"; fi' >> ~/.bashrc

➜ ~ source ~/.bashrc

Una vez instalado RBENV vamos a proceder a instalar la versión de ruby 2.7.1 y a establecerla por defecto

➜ ~ **rbenv** **install** 2.7.1

➜ ~ **rbenv** **global** 2.7.1

Ahora, vamos a configurar GIT usando nuestro nombre y cuenta de correo habitual

$ git config --**global** user.name "Johan Tique"

$ git config --**global** user.email johan@platzi.com

$ git config --**global** color.ui **true**

Para poder mantener los espacios de librerías de forma consistente entre versiones y manifiestos usaremos BUNDLER, la cual es una librería que nos permitirá gestionar a su vez un conjunto de librerías de ruby agrupadas por entornos de trabajo y garantizando la consistencia de versiones y dependencias entre las mismas.

En Ruby, las librerías son llamadas gemas, y bundler usa un archivo manifiesto llamado Gemfile para listar todas las gemas que un proyecto tendrá, bundler puede usar varias fuentes para encontrar de forma pública estas librerías en la nube, sin embargo, nosotros vamos a usar dos fuentes rubygems y github.

Para instalar Bundler (que también es una gema) usaremos el comando gem que es proveído en este caso por RBENV. También habilitaremos el protocolo seguro de HTTP para que cuando se tome github como fuente este sea establecido por defecto.

$ gem **install bundler**

$ **bundle** config github.https true

Rails necesita de un motor de javascript para abordar ciertas funcionalidades de compilación, transformación y en sus últimas versiones, funcionalidades de integración con webpack, es por esta razón que vamos a usar **nodejs** como nuestro motor, sin embargo, NodeJS podría tener la misma situación de conflicto de versiones entre ambientes, por lo que que usaremos también un gestion de versiones de NodeJS para lidiar con estos escenarios.

**NOTA** si ya tienes instalado NodeJS u otro motor de javascript compatible con el intérprete de javascript V8 o similar, no deberás hacer este paso, aunque es ampliamente recomendable usar un gestor de versiones, si lo quieres llegar a usar debes desinstalar el intérprete de javascript que tengas instalado para evitar conflictos futuros.

Para gestionar las versiones de NodeJS usaremos NODENV ejecutando la siguiente secuencia de comandos, que instalará el NODENV, añadirá sus binarios a la variable de entorno PATH, e instalará la versión 12.17.0 de NodeJS.

$ brew install nodenv

$ echo 'eval "$(nodenv init -)"' >> ~/.bashrc

$ source ~/.bashrc

$ nodenv install 12.17.0

$ nodenv global 12.17.0

**Instalando el ambiente de desarrollo con Rails y PostgreSQL**

Habiendo instalado RBENV y BUNDLER ya podemos empezar a instalar todas las gemas, herramientas y servidores necesarios para poder seguir adelante con nuestros primeros pasos con Ruby on Rails

Vamos a instalar Ruby on Rails usando el siguiente comando (si, Rails es también una gema)

$ gem **install** rails

Rails instalará con varias gemas como dependencias para su ejecución, así que obtendremos una instalación con una larga lista de librerías, al final podrás corroborar la versión de Rails usando el comando de abajo, que en nuestro caso será la versión 6.0.3.2

$ rails -v

#=> Rails 6.0.3.2

Las versiones de Rails superiores a la 6, requieren usar YARN como gestor de paquetes de javascript y habilitar el enlace de Webpack usando la tecnología Webpacker, para esto, vamos a usar nuevamente Homebrew

$ **brew install** yarn

Finalmente, vamos a instalar la base de datos de postgreSQL y configurar un usuario para poder gestionarla

$ **brew install** postgresql

Una vez instalado iniciaremos el servicio de la base de datos

$ **brew** services start postgresql

Una vez inicializado postgreSQL, entraremos a la sesión del mismo y crearemos desde allí un nuevo usuario con capacidad de crear bases de datos (de usuario **platzi**, rol **platzi** y contraseña **platzi**), y que usaremos para nuestras futuras aplicaciones, para hacerlo debemos seguir el siguiente conjunto de instrucciones (**No** tener en cuenta el prefijo postgres=# para su ejecución):

$ psql postgres

postgres=# **CREATE** **ROLE** platzi **WITH** LOGIN **PASSWORD** 'platzi';

postgres=# **ALTER** **ROLE** platzi CREATEDB;

Para salir de la sesión de postgreSQl debemos usar el comando \q

**Instalación de Ruby, RoR en Windows 10**

La instalación del entorno de desarrollo en Windows la haremos usando un subsistema de windows para Linux, de esta forma podrás usar una consola tipo Linux que te facilitara la ejecución de instrucciones y hará que tu sistema operativo sea más compatible con la documentación que encuentres en este curso y en la internet.

Como te he mencionado arriba, Windows 10 te permite correr una virtualización nativa de Linux , y usaremos este entorno para instalar todas nuestras herramientas, para preparar nuestro sistema operativo debemos abrir un PowerShell y ejecutalo como administrador como es referenciado en la imagen de abajo.

Una vez allí, ejecuta las siguientes tres instrucciones

dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart

dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart

Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux

Seguido de esto, vamos a abrir el Microsoft Store, buscar el paquete de Ubuntu 20.04 LTS e instalarlo

Una vez instalado, debemos buscar y ejecutar el paquete Ubuntu 20.04 desde el buscador de Windows, al inicio este nos solicitara que creemos un usuario y una nueva contraseña, para nuestro caso será **platzi**

**NOTA IMPORTANTE**

No debes usar el usuario **superusuario** (root) a menos que de forma explícita se especifique que este debe ser usado a través del comando “sudo”, de otra forma no debes usarlo así como tampoco sus variaciones como “$ sudo su”

El programa “sudo” pronunciado “SUDU” /suːduː/ es normalmente interpretado como “superuser do…” y permite al comando inmediatamente siguiente, correr con los permisos de otro usuario que tiene condiciones especiales de seguridad o permisos, comúnmente **sudo** se usa para invocar los permisos del superusuario del sistema operativo, sin embargo, en las últimas versiones de sistemas compatibles con Linux no sólo puede ejecutar como superusuario sino también es posible usar otros tipos especiales de usuario. Sin embargo, nosotros usaremos “sudo” para usar los permisos de superusuario.

**Adecuación del sistema operativo**

**Instalación de librerías base**

Usando sudo y y el comando apt-get (como gestor de paquetes) vas a actualizar los enlaces a los repositorios de las librerías que se pueden instalar en el sistema operativo así como sus dependencias.

$ sudo apt-get **update**

Deberás obtener una respuesta de varias líneas al comando con la actualización de la lista de paquetes y una línea final mencionado que la lectura de las listas de paquetes ha sido finalizada similar a lo siguiente:

Reading **package** **lists**... **Done**

Ahora vamos a instalar las librerias necesarias usando **sudo** y el comando **apt install**

$ sudo apt install build-essential curl wget openssl libssl-dev libreadline-dev dirmngr zlib1g-dev libmagickwand-dev imagemagick-6.q16 libffi-dev libpq-dev cmake libwebp-dev

Para autorizar la instalación deberás aceptar la continuación del proceso digitando la letra **Y** y dando **enter**

**Do** you want **to** **continue**? [Y/n]

Las librerías instaladas cumplirán con los siguientes propósitos:

* **build-essentials:** contiene herramientas para compilar y construir software desde sus fuentes usando generalmente lenguaje C y sus derivados
* **curl:** es una herramienta para transferir información de un servidor a otro usando diversos tipos de protocolos, entre esos: HTTP, FTP, IMAP entre otros
* **wget:** es una herramienta para recibir contenido desde la web usando los protocolos más comunes como HTTP y FTP
* **openssl:** es un robusto conjunto de librerías que te permitirán manipular y leer contenido de los protocolos TLS y SSL, que son usados para garantizar la seguridad y encriptación de las comunicaciones entre servidores y clientes.
* **libssl-dev:** es una librería que forma parte de OpenSSL usada para lidiar con procesos de encriptación, este paquete, contiene librerías de desarrollo, cabeceras de compilación entre otro tipos de archivos.
* **libreadline-dev:** aporta la librerías de desarrollo para el paquete readline que ayuda a la consistencia de interfaces de usuario que están asociadas a líneas interactivas de comandos.
* **dirmngr:** es un pequeño servidor para gestionar y descargar certificados de tipo X.509 y la lista de revocación de los mismos.
* **zlib1g-dev:** aporta librerías de desarrollo para soportar mecanismos de compresión y descompresión compatibles con GZIP y PKZIP, tecnologías de compresión comunes en paquetes de herramientas para nuestro entorno.
* **imagemagick-6.q16:** es un programa que permite editar, crear y componer mapas de bits en imágenes.
* **libmagickwand-dev:** es un conjunto de librerías de desarrollo para compilar librerías necesarias para el uso de MagickWand, este último es la tecnología predilecta como interface del lenguaje de programación C a ImageMagick.
* **libffi-dev:** provee librerías de desarrollo para habilitar mecanismos de alto nivel para el uso de *calling conventions* sobre ciertos compiladores, de esta forma esta librería le permite al desarrollador invocar cualquier función específica por medio de una llamada en tiempo de ejecución.
* **libpq-dev:** esta librería contiene paquetes de desarrollo para habilitar el uso de conjuntos de binarios y cabeceras requeridas para construir componentes externos para PostgreSQL
* **cmake:** es usado para controlar el proceso de compilación de software usando un método independiente del compilador.
* **libwebp-dev:** habilita un conjunto de librerías de desarrollo para el manejo de formatos de imágenes de nueva generación compatibles con WebP

**Instalación de GIT**

A continuación instalaremos nuestro sistema para controlar versiones de nuestro código, para este caso usaremos GIT, y procederemos a su instalación usando el siguiente comando

$ sudo apt **install** git

Una vez instalado GIT configuraremos de forma global algunas variables

$ git config --**global** user.name "Johan Tique"

$ git config --**global** user.email johan@platzi.com

$ git config --**global** color.ui **true**

**Solo si** tu conexión a internet está habilitada a través de un proxy (muy común en las instituciones públicas) debemos configurarlo de la siguiente manera

$ git config --global http.proxy http://proxy.alu.uma.es:3128

$ git config --global https.proxy https://proxy.alu.uma.es:3128

**Instalando RBENV**

Imagina que estás trabajando en varios proyectos con Ruby y Ruby on Rails, y que todos tengan diferentes versiones… **¿cómo haces para garantizar que todas librerias, versiones y paquetes sean consistentes en cada uno de los proyectos?**

Bien, para resolver esto debemos usar gestores de versiones, en este caso usaremos RBENV, no debes confundirte con controladores de versiones (como GIT), son dos tecnologías con propósitos distintos, pero que se complementan.

Para instalar RBENV debemos ejecutar el siguiente comando

$ curl -fsSL https://github.com/rbenv/rbenv-installer/raw/master/bin/rbenv-installer | bash

Al final de la ejecución del comando deberás ver un texto similar a este, mencionando que a pesar de haber instalado RBENV este no se encuentra en la variable de entorno PATH

Running doctor **script** **to** verify installation...

Checking **for** `rbenv' **in** PATH: **not** found

You seem **to** have rbenv installed **in** `/home/platzi/.rbenv/bin', **but** **that**

directory **is** **not** present **in** PATH. Please add **it** **to** PATH **by** configuring

your `~/.bashrc', `~/.bashrc', **or** `~/.config/fish/config.fish'.

Para añadir a la variable de entorno PATH los binarios de RBENV y a su vez ejecutarlo cada vez que exista una nueva sesión, debemos agregar dos líneas al archivo .bashrc que habíamos visto antes, pero esta vez solo usaremos la consola para hacerlo, de la siguiente manera:

$ echo 'export PATH="$HOME/.rbenv/bin:$PATH"' >> ~/.bashrc

$ echo 'eval "$(rbenv init -)"' >> .bashrc

$ source ~/.bashrc

Una vez instalado RBENV vamos a usarlo para instalar la versión 2.7.1 ejecutando lo siguiente (la instalación suele tardar unos minutos, así que asegurate de tener una buena velocidad y estabilidad de internet)

$ rbenv **install** 2.7.1

RBENV está en la capacidad de gestionar un sin número de versiones de ruby así como sus espacios de librerías de forma independiente, así que vamos a establecer la versión 2.7.1 como la versión por defecto y global, usando el siguiente comando.

$ rbenv **global** 2.7.1

Para poder mantener los espacios de librerías de forma consistente entre versiones y manifiestos usaremos BUNDLER, la cual es una librería que nos permitirá gestionar a su vez un conjunto de librerías de ruby agrupadas por entornos de trabajo y garantizando la consistencia de versiones y dependencias entre las mismas.

En Ruby, las librerías son llamadas gemas, y bundler usa un archivo manifiesto llamado Gemfile para listar todas las gemas que un proyecto tendrá, bundler puede usar varias fuentes para encontrar de forma pública estas librerías en la nube, sin embargo, nosotros vamos a usar dos fuentes rubygems y github.

Para instalar Bundler (que también es una gema) usaremos el comando gem que es proveído en este caso por RBENV. También habilitaremos el protocolo seguro de HTTP para que cuando se tome github como fuente este sea establecido por defecto.

$ gem **install bundler**

$ **bundle** config github.https true

Rails necesita de un motor de javascript para abordar ciertas funcionalidades de compilación, transformación y en sus últimas versiones, funcionalidades de integración con webpack, es por esta razón que vamos a usar **nodejs** como nuestro motor, sin embargo, NodeJS podría tener la misma situación de conflicto de versiones entre ambientes, por lo que que usaremos también un gestion de versiones de NodeJS para lidiar con estos escenarios.

**NOTA** si ya tienes instalado NodeJS u otro motor de javascript compatible con el interprete de javascript V8 o similar, no deberás hacer este paso, aunque es ampliamente recomendable usar un gestor de versiones, si lo quieres llegar a usar debes desinstalar el intérprete de javascript que tengas instalado para evitar conflictos futuros.

Para gestionar la versiones de NodeJS usaremos NODENV ejecutando la siguiente secuencia de comandos, que instalará el NODENV, añadirá sus binarios a la variable de entorno PATH, e instalará la versión 12.17.0 de NodeJS.

$ curl -fsSL https://raw.githubusercontent.com/nodenv/nodenv-installer/master/bin/nodenv-installer | bash

$ echo 'export PATH="$HOME/.nodenv/bin:$PATH"' >> ~/.bashrc

$ echo 'eval "$(nodenv init -)"' >> ~/.bashrc

$ source ~/.bashrc

$ nodenv install 12.17.0

$ nodenv global 12.17.0

**Instalando el ambiente de desarrollo con Rails y PostgreSQL**

Habiendo instalado RBENV y BUNDLER ya podemos empezar a instalar todas las gemas, herramientas y servidores necesarios para poder seguir adelante con nuestros primeros pasos con Ruby on Rails

Vamos a instalar Ruby on Rails usando el siguiente comando (si, Rails es también una gema).

$ gem **install** rails

Rails instalará con varias gemas como dependencias para su ejecución, así que obtendremos una instalación con una larga lista de librerías, al final podrás corroborar la versión de Rails usando el comando de abajo, que en nuestro caso será la versión 6.0.3.2

$ rails -v

#=> Rails 6.0.3.2

Las versiones de Rails superiores a la 6, requieren usar YARN como gestor de paquetes de javascript y habilitar el enlace de Webpack usando la tecnología Webpacker, para esto, vamos a usar la versión oficial de los repositorios de YARN a través de los siguientes comandos

$ sudo curl -sS https://**dl**.yarnpkg.**com**/debian/pubkey.gpg | sudo apt-key add -

$ sudo **sh** -**c** "echo 'deb https://dl.yarnpkg.com/debian/ stable main' >> /etc/apt/sources.list"

$ sudo apt **update**

Para instalar YARN vamos a usar el comando de abajo, y usaremos nuestra versión de NodeJS instalada en el paso anterior usando la opción --no-install-recommends

$ sudo apt --no-**install**-recommends **install** yarn

Finalmente, vamos a instalar la base de datos de postgreSQL descargando el ejecutable desde esta pagina <https://www.postgresql.org/download/windows/> recuerda el usuario y la contraseña que asignes en el proceso de instalación, dado que lo necesitaremos para la configuración de Rails.

**Accediendo a mi código**

Para usar el sistema de directorios de windows dentro del ambiente de Linux, debes utilizar la siguiente instrucción que accederá a la raíz de la unidad **C** y a partir de allí ejecutar tus comandos

$ cd /mnt/c

Alli podras crear directorios de trabajo que podrás acceder desde cualquier editor de texto en tu entorno windows, para nuestro curso haremos el directorio coding/platzi

$ mkdir -**p** coding/platzi