0x00. Vagrant

***Zero-day (computing)***

***Una vulnerabilidad de día cero (también conocida como día 0) es una vulnerabilidad de software de computadora que es desconocida para aquellos que deberían estar interesados en mitigar la vulnerabilidad (incluido el proveedor del software de destino). Hasta que se mitigue la vulnerabilidad, los piratas informáticos pueden explotarla para afectar negativamente a programas informáticos, datos, equipos adicionales o una red. Un exploit dirigido a un día cero se denomina exploit de día cero o ataque de día cero.***

El término "día cero" se refería originalmente al número de días desde que se lanzó al público una nueva pieza de software, por lo que el software de "día cero" era software que se había obtenido pirateando la computadora de un desarrollador antes de su lanzamiento. ***Finalmente, el término se aplicó a las vulnerabilidades que permitieron este pirateo y a la cantidad de días que el proveedor ha tenido que corregirlas. Una vez que el proveedor se entera de la vulnerabilidad, generalmente creará parches o recomendará soluciones para mitigarla.***

***Virtual machine***

***En informática, una máquina virtual (VM) es la virtualización / emulación de un sistema informático. Las máquinas virtuales se basan en arquitecturas informáticas y proporcionan la funcionalidad de una computadora física. Sus implementaciones pueden involucrar hardware, software especializado o una combinación.***

Las máquinas virtuales difieren y están organizadas por su función, que se muestra aquí:

Las máquinas virtuales del sistema (también denominadas máquinas virtuales de virtualización completa) proporcionan un sustituto de una máquina real. Proporcionan la funcionalidad necesaria para ejecutar sistemas operativos completos. Un hipervisor utiliza la ejecución nativa para compartir y administrar hardware, lo que permite múltiples entornos que están aislados entre sí, pero que existen en la misma máquina física. Los hipervisores modernos utilizan virtualización asistida por hardware, hardware específico de virtualización, principalmente de las CPU del host.

Las máquinas virtuales de proceso están diseñadas para ejecutar programas informáticos en un entorno independiente de la plataforma.

Algunos emuladores de máquinas virtuales, como QEMU, están diseñados para emular también (o "imitar virtualmente") diferentes arquitecturas de sistemas, lo que permite la ejecución de aplicaciones de software y sistemas operativos escritos para otra CPU o arquitectura. La virtualización a nivel de sistema operativo permite particionar los recursos de una computadora a través del kernel. Los términos no son universalmente intercambiables.

***¿Por qué utilizar máquinas virtuales? ¿Y por qué Vagrant?***

¡Nos alegra que se esté adaptando al marco de Holberton! ¡Siempre pregunta por qué!

Es importante preguntarse: “¿Por qué no estoy desarrollando solo en mi computadora? ¡Tengo todas las herramientas que necesito! ”.

***Mi máquina frente a entornos virtuales***

El entorno de tu computadora, ya sea Windows, MacOS o una distribución de Linux, cambiará mucho con el tiempo, sin que te des cuenta. Instalarás aplicaciones, juegos, herramientas,… que requerirán e instalarán diferentes dependencias y al final del día puedes terminar teniendo comportamientos completamente diferentes o incluso tener algo que no funcione por conflictos de software.

No entraremos en los detalles de las máquinas virtuales, pero como su nombre lo indica, son "computadoras" virtuales que emularán todo, desde la CPU hasta la RAM y el disco. Las máquinas virtuales en el contexto del desarrollo son un medio para aislar y mantener un entorno estable que básicamente se ejecutará de la misma manera en cualquier host (cualquier computadora). De esta manera, puede tener cualquier software instalado en su Windows, MacOS o cualquier distribución de Linux; su Máquina Virtual ejecutará su propio entorno, tendrá sus propios programas, con sus propias versiones, etc.

El uso de entornos virtuales evita que los desarrolladores digan "No entiendo, funciona en mi máquina ...". Aquí, en Holberton, queremos asegurarnos de que tenga acceso al mismo entorno que se utilizará para corregir su trabajo (el Verificador).

***Las herramientas***

Existen múltiples herramientas que pueden ayudarlo a crear y administrar entornos virtuales (observe que usamos el término “entorno virtual” aquí, ya que dicho entorno no es necesariamente una máquina virtual. Las tecnologías que utilizan la contenedorización permiten administrar entornos virtuales también).

***Estamos utilizando dos herramientas en Holberton: VirtualBox y Vagrant.***

VirtualBox es un proveedor de máquinas virtuales. Las propias máquinas virtuales se generarán mediante VirtualBox. VirtualBox es gratis y liviano, lo que lo convierte en una opción perfecta para nosotros.

Vagrant es una herramienta que se encuentra encima de un proveedor de VM. Nuevamente, elegimos usar VirtualBox como proveedor, pero existen otros proveedores y Vagrant ofrece la posibilidad de usar diferentes proveedores (más información aquí). Al igual que VirtualBox, elegimos usar Vagrant porque es gratuito, confiable y está bien mantenido. Tenga en cuenta que el propósito aquí es usar entornos virtuales, y tanto VirtualBox como Vagrant son solo medios para lograr este propósito.

***Alternativas***

Como se mencionó anteriormente, usar VirtualBox y Vagrant en Holberton no significa que todas las empresas vayan a trabajar exactamente de la misma manera. Se utilizan diferentes herramientas, así como diferentes flujos de trabajo de desarrollo. Queremos que comprenda lo importante que es aislar su entorno de desarrollo de cualquier máquina host, ya sea su computadora personal o una de las computadoras de la escuela.

A continuación, se muestran algunos ejemplos de otros softwares que se utilizan ampliamente para administrar entornos virtuales: The VMWare products & Docker

***Si está utilizando su propia computadora, instale vagrant antes de continuar.***

***Más sobre Vagrant***

Vagrant proporciona entornos de trabajo portátiles, reproducibles y fáciles de configurar, construidos sobre la tecnología estándar de la industria y controlados por un único flujo de trabajo consistente para ayudar a maximizar la productividad y la flexibilidad de usted y su equipo.

***Vagrant es una herramienta para crear y administrar entornos de máquinas virtuales en un solo flujo de trabajo. Con un flujo de trabajo fácil de usar y un enfoque en la automatización, Vagrant reduce el tiempo de configuración del entorno de desarrollo, aumenta la paridad de producción y hace que los "trabajos en mi máquina" sean una reliquia del pasado.***

Si ya está familiarizado con los conceptos básicos de Vagrant, la documentación proporciona una mejor compilación de referencia para todas las funciones y funciones internas disponibles.

***¿Por qué Vagrant?***

Vagrant proporciona entornos de trabajo portátiles, reproducibles y fáciles de configurar, construidos sobre la tecnología estándar de la industria y controlados por un único flujo de trabajo consistente para ayudar a maximizar la productividad y la flexibilidad de usted y su equipo.

Para lograr su magia, Vagrant se coloca sobre los hombros de gigantes. Las máquinas se aprovisionan además de VirtualBox, VMware, AWS o cualquier otro proveedor. Luego, las herramientas de aprovisionamiento estándar de la industria, como scripts de shell, Chef o Puppet, pueden instalar y configurar automáticamente el software en la máquina virtual. ***Vagrant fue escrita por HashiCorp y su relación con las maquinas virtuales es que las administra.***

***Ubuntu***

Es un [sistema operativo](https://es.wikipedia.org/wiki/Sistema_operativo) de [software libre y código abierto](https://es.wikipedia.org/wiki/Software_libre_y_de_c%C3%B3digo_abierto). Es una [distribución de Linux](https://es.wikipedia.org/wiki/Distribuci%C3%B3n_Linux) basada en [Debian](https://es.wikipedia.org/wiki/Debian). Puede correr en [computadores de escritorio](https://es.wikipedia.org/wiki/Computadora_de_escritorio) y [servidores](https://es.wikipedia.org/wiki/Servidor). Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario. Está compuesto de múltiple [software](https://es.wikipedia.org/wiki/Software) normalmente distribuido bajo una [licencia](https://es.wikipedia.org/wiki/Licencia_de_software) [libre](https://es.wikipedia.org/wiki/Software_libre) o de [código abierto](https://es.wikipedia.org/wiki/C%C3%B3digo_abierto). Estadísticas web sugieren que la cuota de mercado de Ubuntu dentro de las distribuciones Linux es, aproximadamente, del 52 %,[3](https://es.wikipedia.org/wiki/Ubuntu#cite_note-3)​[4](https://es.wikipedia.org/wiki/Ubuntu#cite_note-4)​ y con una tendencia a aumentar como servidor web.

***Qué es significa la palabra Ubuntu?***

Ubuntu es una regla ética o filosofía sudafricana enfocada a la lealtad con las personas y a su manera de relacionarse. La palabra proviene de las lenguas Zulú y Xhosa. Surge del dicho popular «Umuntu, nigumuntu, nagumuntu» que en Zulú significa « una persona es una persona a causa de los demás».

***Qué hace el comando “uname”?***

***imprime la información del sistema.***

***¿Qué es la gestión del código fuente?***

***Permite a un desarrollador organizar las iteraciones de código cronológicamente y versionarlo para una aplicación. Las características más poderosas de los sistemas de administración de código fuente se encuentran en cómo permiten que equipos de muy diversos tamaños trabajen juntos en la misma aplicación simultáneamente.***

***Algunos términos que son comunes a todos ellos:***

Un proyecto se llamará repositorio, representa el índice / la raíz del sistema de archivos de su proyecto.

Los desarrolladores pueden crear ramas del código base, en las que iterarán por separado para otros desarrolladores. Por ejemplo, una rama puede estar destinada a utilizarse para una función determinada o para corregir un error. Se pueden crear tantas ramas como necesiten.

Una vez que una rama está lista para ello (ha sido probada, revisada por pares, etc.), se puede fusionar nuevamente con la rama principal. La rama principal se puede llamar de manera diferente: en Git se llama master, en SVN se llama trunk.

Mientras codifican en su rama, los desarrolladores deben trabajar en pequeñas iteraciones atómicas, llamadas confirmaciones. Todas las confirmaciones tienen un mensaje de confirmación que describe en una oración lo que hay allí.

Todas las confirmaciones juntas se denominan historia, y es muy importante escribir confirmaciones significativas y mensajes de confirmación para mantener limpio el historial del proyecto de un vistazo, para comprender qué ha estado sucediendo y quién hizo qué.

Algunas personas pueden estar modificando las mismas piezas en el código base en diferentes ramas, y esto podría crear conflictos cuando uno fusiona esas ramas. Algunos de esos conflictos, obviamente, solo pueden ser solucionados por un humano, y cada sistema tiene una forma diferente de gestionar los conflictos de fusión.

***¿Cuál es la diferencia entre Git y GitHub?***

***Git es todo lo que hemos cubierto hasta ahora: una herramienta de administración de código fuente, que viene con una herramienta de línea de comandos para sus usuarios.***

***GitHub es uno de los muchos servicios que brindan al mismo tiempo: un servidor de repositorio de Git para enviar su código a una interfaz de usuario web para ver sus repositorios, con sus archivos y confirmar una serie de características adicionales (administrar su equipo y accesos, ...)***

***Git, ante todo, es una herramienta de control de versiones.***

***Dos características de GitHub con las que debe familiarizarse son:***

***Bifurcaciones:*** puede bifurcar cualquier repositorio en GitHub, y duplicará el código base del repositorio en el repositorio de su propiedad. Por ejemplo, si bifurca twbs / bootstrap y tu nombre de usuario de GitHub es "my\_username", entonces creará el repositorio my\_username / bootstrap, y recordará desde dónde se bifurcó. Por lo general, no se le permite presionar en los repositorios de otras personas, por lo que le proporcionará un repositorio al que puede presionar, ya que es el propietario.

***Solicitudes de extracción:*** una vez que haya enviado su código a su repositorio (o, a veces, a una rama del repositorio principal, si está permitido), puede crear una solicitud de extracción hacia la rama principal del repositorio principal. Alguien a cargo del repositorio principal revisará su solicitud de extracción (posiblemente le pida que cambie un par de cosas) y la fusionará si es adecuado para estar en el producto principal.

*GitHub tiene muchos competidores, dos de los principales son GitLab y BitBucket (que brindan servicios muy similares). En Holberton School, decidimos hacerte usar GitHub porque ahí es donde está la mayor parte de la industria (de esa manera, podrás interactuar con ellos en sus proyectos de código abierto), y también es donde los reclutadores de tecnología suelen visitar. para ver lo que has estado haciendo.*

***Sobre README***

***Puede agregar un archivo README a un repositorio para comunicar información importante sobre su proyecto. Un archivo README, junto con una licencia de repositorio, pautas de contribución y un código de conducta, comunica las expectativas de su proyecto y lo ayuda a administrar las contribuciones.***

*Un archivo README es a menudo el primer elemento que verá un visitante cuando visite su repositorio. Los archivos README suelen incluir información sobre:*

*Que hace el proyecto.*

*Por qué el proyecto es útil.*

*Cómo los usuarios pueden comenzar con el proyecto.*

*Dónde los usuarios pueden obtener ayuda con su proyecto.*

*Quién mantiene y contribuye al proyecto.*

Si coloca su archivo README en la raíz de su repositorio docs, o en el .githubdirectorio oculto , GitHub reconocerá y mostrará automáticamente su archivo README a los visitantes del repositorio.

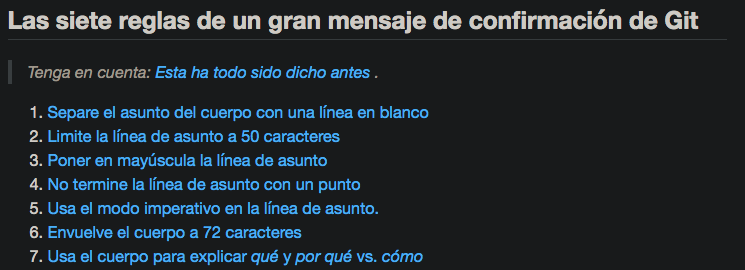
***Cómo escribir mensajes de confirmación útiles?***

Tienen que ser concisos y coherentes.  
Lo primero es lo que ocurre por defecto; esto último nunca ocurre por accidente.

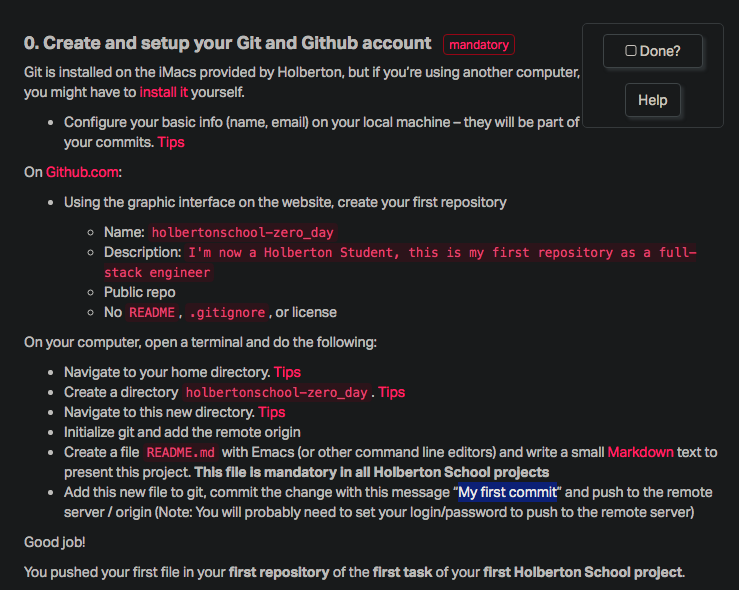
Si bien los registros de muchos repositorios se parecen a los anteriores, existen excepciones. El [kernel de Linux](https://github.com/torvalds/linux/commits/master) y [Git en sí](https://github.com/git/git/commits/master) son excelentes ejemplos. Mire [Spring Boot](https://github.com/spring-projects/spring-boot/commits/master) o cualquier repositorio administrado por [Tim Pope](https://github.com/tpope/vim-pathogen/commits/master) .

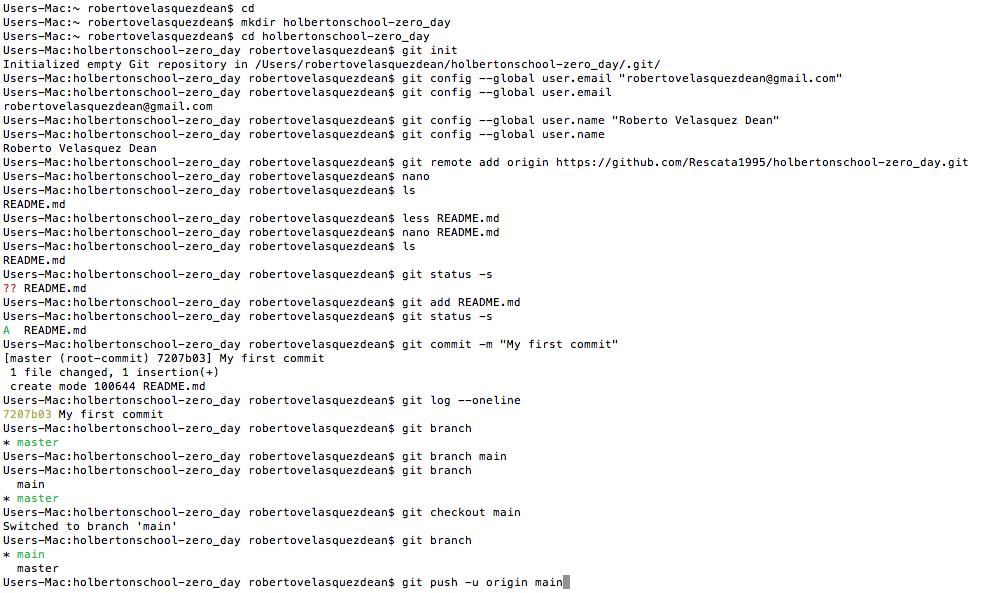
Los colaboradores de estos repositorios saben que un mensaje de confirmación de Git bien elaborado es la mejor manera de comunicar el contexto sobre un cambio a los compañeros desarrolladores (y de hecho a sus futuros yo). Una diferencia le dirá qué cambió, pero solo el mensaje de confirmación puede decirle correctamente por qué . Peter Hutterer [hace bien este punto](http://who-t.blogspot.co.at/2009/12/on-commit-messages.html) :

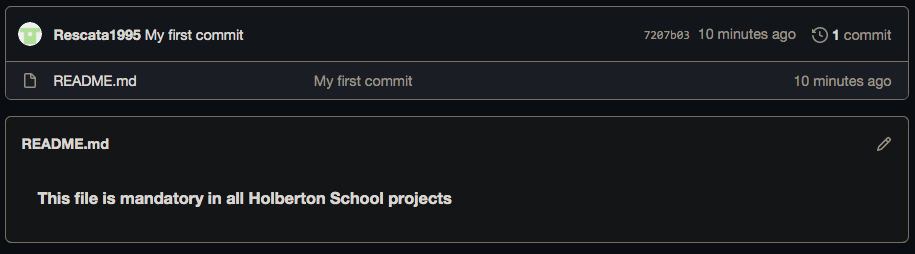
Restablecer el contexto de un fragmento de código es un desperdicio. No podemos evitarlo por completo, por lo que nuestros esfuerzos deben dirigirse a [reducirlo](http://www.osnews.com/story/19266/WTFs_m) [tanto] como sea posible. Los mensajes de confirmación pueden hacer exactamente eso y, como resultado, un mensaje de confirmación muestra si un desarrollador es un buen colaborador.



Solución punto 1 - proyecto 0







Solución punto 2 - proyecto 0



