

final

Infraestructura I

Índice

1. Inmersion

- C1 - Introducción a la materia & the big picture
 - Bienvenida
 - Introducción a la materia
 - El centro de computadoras
 - The big picture

2. Automatizacion

- C2 - Automatizacion
 - Automatizacion de la infraestructura
 - 5 Tareas mas comunes de automatizar
 - Configuracion y mantenimiento del sistema
 - Infraestructura y servicios
 - Manejo del codigo
 - Contenedores
 - Ambientes de trabajo
 - Monitores de red
 - Lenguajes de scripting
 - Virtualizacion
 - Historia virtualizacion
 - Componentes virtualizacion
 - Instalacion VM Debian, Apache2, OpenSSH, puTTy
- C3 - Repaso
 - Vagrant: box Debian
 - Automatizar: modificando file de vagrant
 - Cambiar HTML base modificando file vagrant
- C4 - Shell Scripting - Parte I
 - Introducción a la terminal de Linux
 - La consola de Linux
 - Tipos de shell
 - Bourne Shell
 - C/TC Shell
 - Korn Shell
 - Bourne-Again Shell (BASH)
 - Ejecucion de la consola
 - Elevacion de privilegios
 - Comandos más utilizados en la terminal de Linux
 - Consolidando nuestro ambiente
 - Verificando nuestro ambiente

- Comandos para el manejo de archivos
- Comandos para leer archivos de texto
- Obtener datos desde un web service
- Comando cURL
- Comando jq
- Combinación de uso de ambos comandos
- Clase en vivo 4
 - Listar servicios del sistema
 - Analizando los datos que van y vienen
- C5 - Shell Scripting - Parte II
 - Scripting
 - Bash scripting
 - Estructuras de control
 - Sentencia if-then
 - Sentencia if-else
 - Comparaciones numéricas
 - Comparaciones de cadenas
 - Cálculos matemáticos
 - Ejemplo en Bash
 - Notas clase 5
- C6 - Repaso
- C7 - PowerShell
 - Introducción a PowerShell
 - Diferencias entre PowerShell y PowerShell Core
 - Usos comunes de PowerShell
 - Comandos cmdlet
 - Ejemplo uso comandos cmdlet
 - Kit de supervivencia de PowerShell
 - módulos de PowerShell
 - Notas clase 7
- C8 - Python
 - Introducción a Python
 - Importancia de python en infraestructura
 - Pros y contras de Python
 - Nos ponemos a practicar
 - Variables
 - Tipos de variables
 - Indentación (sangría)
 - if en Python
 - Condiciones en Python
 - Bucles while y for
 - Ejemplos estructuras de control
 - instrucciones de selección
 - estructuras de selección
- C10 - Configuration Management
 - Change management

3. Containers
4. Cloud computing
5. Cierre

C1 - Introducción a la materia & the big picture

Modulos infraestructura

Introducción a la materia

En la actualidad, la actividad cotidiana depende en un porcentaje elevado del correcto funcionamiento de los sistemas informáticos —y saber operar con dichos sistemas—. La dependencia del ámbito productivo con los complicados sistemas informáticos es tal, que un problema en una terminal de trabajo deja prácticamente sin efecto la capacidad operativa de cualquier usuario. Ante situaciones de esta naturaleza —considerando la complejidad de las actuales plataformas tecnológicas y la multiplicidad de usuarios con diversos niveles de acceso y experticia—, se hace necesaria la presencia de un profesional confiable que intervenga, rápida y eficazmente, con habilidades y destrezas mayores a las del operador habitual.

La función de este profesional —conocido como administrador de infraestructura— es brindar respuestas a los problemas habituales de hardware y software en cualquier ámbito informatizado, planteándose desde una problemática y evaluando su mejor solución. Este servicio debe estar en manos de profesionales que orienten, configuren, prevengan y resuelvan eventualidades, manteniendo sus equipos en un estado óptimo. Además, el administrador de infraestructura debe ser capaz de prestar servicios de administración y soporte de sistemas de base y elementos de infraestructura para el procesamiento de aplicaciones informáticas, tales como servidores, dispositivos de almacenamiento masivo, otros dispositivos de hardware, sistemas operativos, máquinas virtuales y administradores de redes, servicios de comunicaciones a través de redes públicas y privadas, dispositivos de switching, firewalls, motores de bases de datos, entre otros. Podrá además brindar servicios de administración de la infraestructura tecnológica en la cual opera el software de estas aplicaciones interviniendo en forma puntual para resolver los problemas que experimente esa infraestructura o su eficiencia operativa y realizar un diagnóstico de incidentes que se presenten en la operatoria habitual del sistema.

La intención es contribuir con soluciones concretas a la problemática del soporte de infraestructura IT que se presentan en la casi totalidad de las instituciones públicas y/o privadas, acelerando los tiempos en materia de formación de recursos humanos capacitados y demandados por todas aquellas organizaciones que cuentan con una infraestructura IT.

¿Qué es infraestructura de IT?

La infraestructura de tecnología de la información, o infraestructura de IT, se refiere a los componentes combinados necesarios para el funcionamiento y la gestión de los servicios de IT de la empresa y los entornos de IT.

¿Quién hace infraestructura?

Estudiar una carrera profesional no consiste solamente en los años que le vas a dedicar a terminar tus estudios. Despues de obtener tu título te espera un largo camino de crecimiento profesional y, si bien muchas

veces tenemos una idea de las opciones profesionales que existen, a veces tenemos nociones erradas o no hemos explorado correctamente todas las posibilidades que hay disponibles. Por eso, a lo largo de la carrera te compartiremos las opciones en el campo laboral que tienen todos aquellos que decidan ser profesionales de sistemas.

Es evidente que estamos en una era en la que casi cualquier tipo de tarea es mediada por un computador. Por eso, el conocimiento de un profesional de sistemas es necesario en sectores tan diversos como, por ejemplo, el entretenimiento o el mundo empresarial. A través de esta materia adquirirán conocimientos aplicables a las tareas que deben desarrollar en cualquier sector de IT.

¿Cuál es el perfil de un analista de infraestructura?

Las funciones que ejerce un analista de infraestructura, entre otras, son:

- Administrar servidores, software de base, comunicaciones y demás subsistemas, maximizando el aprovechamiento de los recursos y anticipando posibles eventualidades.
- Administrar redes de comunicación de datos (cableadas o no), asegurando la accesibilidad de los servicios y optimizando los recursos.
- Atender incidentes que afecten al soporte de infraestructura de IT, diagnosticar las causas que los originan y resolverlos o coordinar su solución.
- Instalar o reemplazar componentes de soporte de infraestructura de IT o adaptarla a nuevas condiciones de servicios externos minimizando riesgos para la seguridad y continuidad del servicio.
- Migrar o convertir sistemas, aplicaciones o datos tratando de minimizar riesgos para la seguridad y continuidad del servicio.
- Entender temas de contingencias y riesgos que puedan afectar al soporte de infraestructura de IT.
- Generar propuestas innovadoras y/o emprendimientos productivos propios del ámbito de la gestión de soporte de infraestructura de IT.

Esta persona se desempeña en centros de procesamiento de datos, ya sean de empresas u organizaciones de cualquier tipo usuarias de tecnología de la información. Su posición ocupacional suele denominarse administrador de red o administrador de sistemas, y trabaja solo o en pequeños grupos para administrar los recursos de infraestructura de tecnología de la información y atender y resolver incidentes, a fin de minimizar la posibilidad de interrupciones al servicio que brindan las aplicaciones informáticas a las organizaciones.

Por lo general, depende directa o indirectamente de un gerente de tecnología responsable por toda la operación y —en función de la dimensión de la organización en la cual se desempeñe— puede trabajar solo, en pequeños grupos o en grupos más grandes que permitan su especialización en determinadas tecnologías. En la mayoría de los casos, no tiene personal a cargo, aunque puede coordinar las actividades de pequeños grupos operativos. En algunos casos en que la infraestructura es muy pequeña puede brindarle sus servicios profesionales externamente atendiendo a los centros de procesamiento en forma presencial o a distancia.

¿Por qué es tan importante la infraestructura de IT?

La tecnología es el motor de prácticamente todos los aspectos de las empresas actuales. Desde el trabajo de un empleado individual hasta las operaciones de bienes y servicios. Si se conecta correctamente, la tecnología se puede optimizar para mejorar la comunicación, crear eficiencia y aumentar la productividad.

Si una infraestructura de IT es flexible, fiable y segura, puede ayudar a una empresa a cumplir sus objetivos y a ofrecer una ventaja competitiva en el mercado. De forma alternativa, si una infraestructura de IT no se

implementa correctamente, las empresas pueden afrontar problemas de conectividad, productividad y seguridad, como infracciones e interrupciones del sistema. En general, contar con una infraestructura correctamente implementada puede ser un factor clave para la rentabilidad de un negocio.

Con una infraestructura de IT, una empresa puede:

- Ofrecer una experiencia del cliente positiva proporcionando acceso ininterrumpido a su sitio web y la tienda en línea.
 - Desarrollar y lanzar soluciones al mercado con rapidez.
 - Recopilar datos en tiempo real para tomar decisiones rápidas.
 - Mejorar la productividad de los empleados.
-

Arquitectura cliente-servidor

La arquitectura cliente-servidor persigue el objetivo de procesar la información de un modo distribuido. De esta forma, pueden estar dispersos en distintos lugares y acceder a recursos compartidos.

Además de la transparencia y la independencia del hardware y del software, una implementación cliente-servidor debe tener las siguientes características:

- Utilizar protocolos *asimétricos*, donde el servidor se limita a escuchar en espera de que un cliente inicie una solicitud.
- El acceso es transparente, multiplataforma y multiarquitectura.
- Se facilitará la *escalabilidad*, de manera que sea fácil añadir nuevos clientes a la infraestructura —escalabilidad horizontal— o aumentar la potencia del servidor o servidores, aumentando su número o su capacidad de cálculo —escalabilidad vertical—.

Ahora veamos las características de los tres componentes de la arquitectura cliente-servidor.

Arquitectura cliente-servidor

Ver video: La importancia del centro de datos

dentro de los middleware estan:

DBMS

Sistema de administracion de base de datos que permiten:

- crear
- Recuperar
- Actualizar
- Administrar datos

APIs

Interfaz de programacion de aplicaciones que brindan un conjunto de subrutinas funciones y procedimientos para ser utilizados en otro software

Sistema operativo

El motor que va a aprovechar los recursos de la infraestructura, conjunto de programas capaz de administrar los recursos físicos del servidor así como los protocolos de ejecución de otro software

Características fundamentales de un sistema operativo

Suporte de red

Es indispensable que tengan un soporte completo para poder brindar conectividad.

Amplia compatibilidad con el hardware

Un punto fundamental para el aprovechamiento pleno de las características del servidor es que el S.O. sea capaz de exprimir al máximo las características técnicas del hardware en donde se ejecuta, es por ello que se debe priorizar el uso de S.O. actualizados y con un soporte importante de controladores. Por ejemplo, que nuestros controladores permitan acceder a características avanzadas de gestión de discos rígidos, con el propósito de realizar arreglos redundantes para tener mayor velocidad y tolerancia a fallos.

Seguridad

Teniendo en cuenta el rol que cumple el servidor, es de vital importancia que el S.O. instalado sea seguro; eso implica no solo que este actualizado con todos los parches/actualizaciones, sino que además debe tener aplicadas políticas estrictas de acceso para prevenir accesos no autorizados o ataques.

Adicionalmente esa seguridad debe reforzarse con la instalación de Firewalls (por software o hardware) y antivirus.

En este ítem también debemos incluir el respaldo de la información, ya sea por medio de herramientas que el propio S.O. nos ofrezca o instalando software externo, con el propósito de tener la menor pérdida de datos posible en caso de fallos fatales.

Dispositivos físicos de protección

Debemos priorizar S.O. que en su arquitectura permita una tolerancia a fallos, ya sea mediante la generación de granja de servidores, que interconectados, operen como una gran unidad de proceso, dando la posibilidad que ante la caída de uno de los integrantes de la granja, otro puede tomar su rol y responsabilidad.

Conocidas estas características casi innegociables a la hora de elegir un S.O. para servidores, es común preguntarnos por qué se utilizan sistemas operativos y versiones específicas para el entorno de servidores y no usamos los mismos S.O. que tenemos instalados habitualmente en nuestras computadoras o estaciones de trabajo; eso se basa en algunas diferencias, a destacar:

Manejo diferente del hardware:

Debido al diferente propósito, los S.O. de estaciones de trabajo no pueden aprovechar todo el hardware disponible, como, por ejemplo, el manejo de memoria RAM —teniendo el caso de Windows 10 64-bit que puede manejar 6TB de RAM, mientras que Windows Server 2019 alcanza los 24TB—.

Características soportadas:

Hay funcionalidades que nativamente un S.O. de estación de trabajo no es capaz de brindar, ya que en su versión de kernel están limitadas o deshabilitadas —casos tales como la virtualización en algunas versiones de

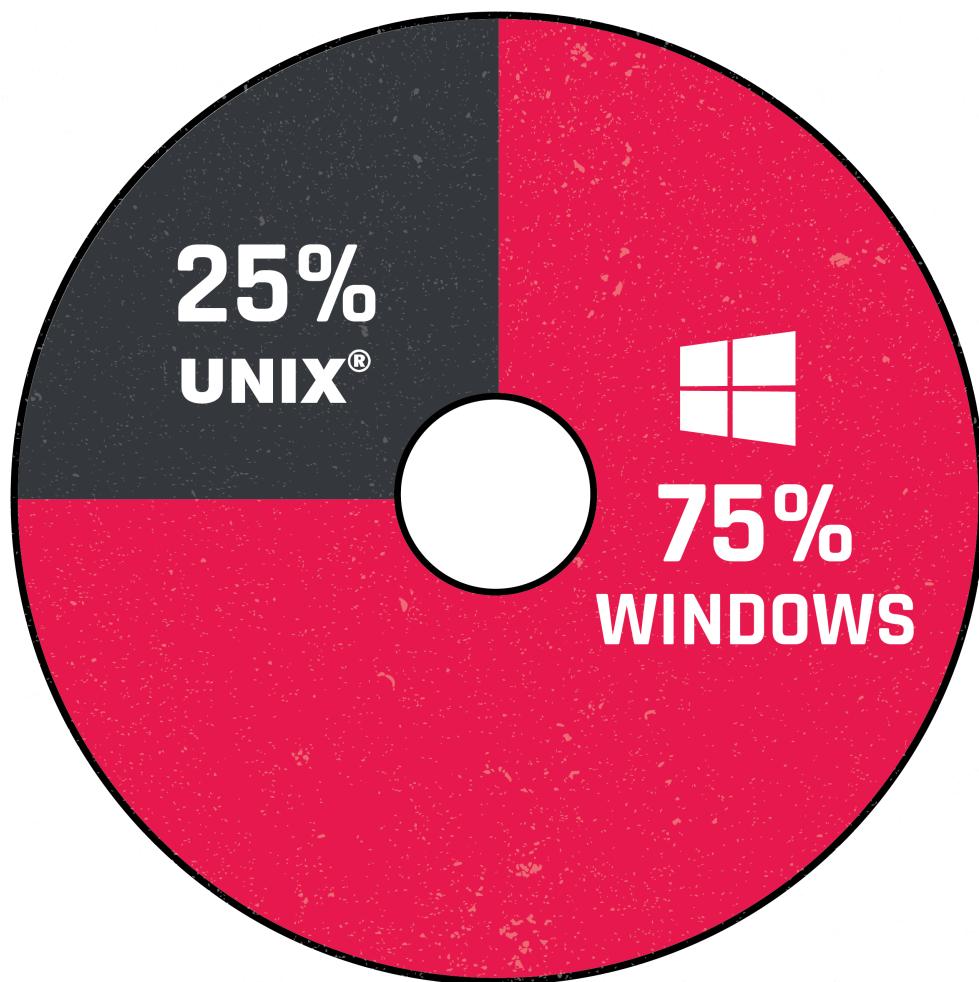
Windows 10—.

Soporte:

Algo muy importante a tener en cuenta es, cuando nuestro negocio o aplicación depende de un S.O., es el **soporte por parte del fabricante/desarrollador**. En el caso de los S.O. de estación de trabajo, el soporte/cobertura que tenemos es para un uso específico, **si sobre esa base quisieramos desplegar una arquitectura, por ejemplo, de servidor web, si bien es probable que nos funcione, vamos a carecer de soporte técnico** ya que el fabricante nos indicará que para ese propósito esta la versión "Server" del producto.

Ya entrando en el terreno de las opciones disponibles para un S.O. de servidor, nos vamos a encontrar con dos opciones predominantes en el mercado:

Utilización de S.O. en Servidores Web



En el grupo de los UNIX (o UNIX-like) tenemos a:

- GNU Linux
- FreeBSD
- macOsServer

En el grupo de Windows, integrado por toda la familia de Windows Server, tenemos las versiones que van desde la versión 2003 a 2019.

En el transcurso de la cursada nos vamos a centrar en GNU Linux y Windows, ya que son las opciones más utilizadas; iremos conociendo algunas características que ofrecen, como así también sus diferencias.

juego unir conceptos

Arquitectura cliente-servidor

Unir los conceptos con sus definiciones.

Cliente	Realiza peticiones a otro programa, el servidor, quien le da respuesta.
Servidor	Es un conjunto de computadoras capaz de atender las peticiones de un cliente y devolverle una respuesta en concordancia.
DBMS	Es un sistema de administración de base de datos que permite crear, recuperar, actualizar y administrar datos.
API	Es una interfaz de programación de aplicaciones que brindan un conjunto de subrutinas, funciones y procedimientos para ser utilizados por otro software.
Respuesta	Una vez que nuestra petición fue recibida por el servidor, este se va a encargar de procesarla y en función de su lógica, la prepara.
Sistema operativo	Es una plataforma que permite ejecutar programas y aplicaciones multiusuario.

¿Querés saber todo lo que vas a aprender en esta materia?

Los profesores te van a dar la bienvenida a esta gran aventura. Comenzarán explicándote cómo se compone y funciona un centro de cómputos. De esta manera, te abrimos la ventana al mundo de la infraestructura.

Te vamos a dar una visión teórica sobre el paradigma cliente-servidor y te vamos a llevar de viaje desde la PC de un cliente hacia el data center. Vas a aprender acerca de sistemas operativos en el contexto del data center. A saber la diferencia entre los distintos sistemas operativos que existen en el mercado. Y, como si fuera poco, vas a saber sobre los usos y servicios que se les pueden dar.

Seguiremos con la historia de la virtualización, analizando las capas de servicios con sus funcionalidades y, sobre todo, sabrán de los beneficios de esta tecnología. Te sumergiremos en el mundo de las redes mostrándote el modelo OSI, descubriendo sus protocolos y te llevaremos de viaje con un paquete de una estación de trabajo a otra. Vas a entender el famoso chiste del que hablan los informáticos sobre capa 8. Te mostraremos el mundo de los procesos ITIL: service request management, incident management, problem

management y change management; por si llegás a tener que ver algún caso en una herramienta de tickets en tu carrera, vas a saber y poder diferenciarlos.

Además, te introduciremos en el maravilloso mundo de la criptografía. Vas a ver cómo puede ser utilizada para autenticar partes, principalmente, en el contexto de SSH y HTTPS. Aprenderás los comandos básicos para el copiado, borrado y creación de archivos y directorios. También te explicaremos los comandos básicos para la obtención de datos desde un web service y posterior proceso.

Estudiarás qué es el pipeline en la consola y te explicaremos Bash como lenguaje de scripting.

Luego, vamos a pasar por PowerShell, otro lenguaje de automation muy utilizado en las empresas hoy en día. Vas a saber cómo instalar, abrir y ejecutar comandos en una consola y los distintos módulos que existen que te van a ayudar a simplificar muchos futuros scripts.

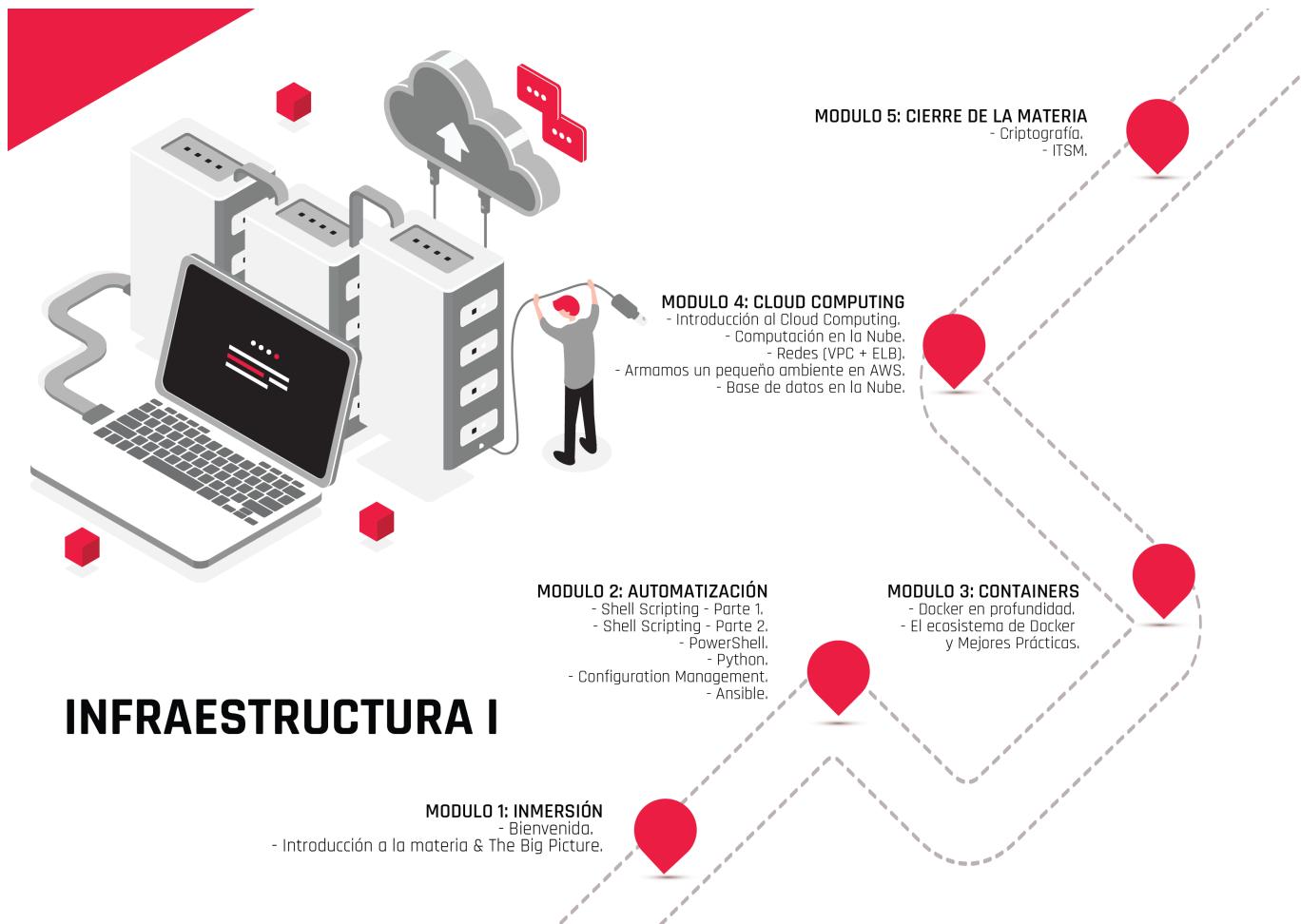
Para que no te quedes afuera de los lenguajes de scripting, vas a ver también Python, uno de los más reconocidos hoy en día. Te vamos a dar una perspectiva de para qué es y para qué sirve y vas a poder reconocer las aplicaciones —que tanto usás en el día a día— que fueron creadas en ese lenguaje.

Seguimos el camino de esta carrera mostrándote configuration management y Ansible. A esta altura vas a usar el emoji con los anteojitos de sol o los de lectura, el que te guste más.

Continuaremos con una tecnología que nos encanta (y a vos seguro también): Docker. Recorreremos todos los productos de esta simpática ballenita y vamos a ver todos sus usos y mejores prácticas.

Terminaremos este viaje que es Infraestructura I mostrándote todo lo relacionado a cloud computing. Analizaremos los modelos de responsabilidad y vas a poder diferenciar los modelos IaaS, PaaS y SaaS. Conocerás modelos como nube híbrida y serverless. Te mostraremos diferentes nubes: Azure, Google y AWS. Practicarás con AWS Educate y armaremos un pequeño ambiente.

Del otro lado somos un equipo que armamos con mucha pasión todo el contenido de esta materia, con los requerimientos que el programa pide y de lo que nos hubiese gustado aprender a nosotros en el paso que ahora están haciendo. Te agradecemos enormemente tu tiempo y esperamos que nuestro aporte te sea de mucha utilidad en tu carrera. ¿Comenzamos?



Mesa nº 7

Ospina, Liliana; Mirra, Benjamín; Groisman, Carina Judith; Valencia Castaño, Luisa María; Valencia, Ronald Nolberto David

Ejercicio nº 1

*. Escenario 1:

- servicio de hosting: apache
- servidor de base de datos: MySQL
- Presupuesto escaso: Nombre de dominio

*. Escenario 3:

- Sistema operativo: Centos
- Presupuesto holgado:
- Amazon AWS

*. Escenario 5:

- Servidor base de datos: MongoDB
- Sistema operativo : CentOS
- Servidor web: Apache

*. Escenario 2:

- Servidor: windows server
(maneja active directory)

*. Escenario 4:

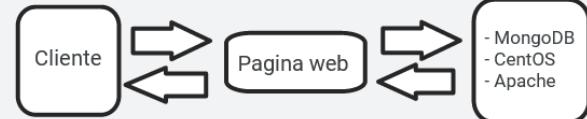
- Servidor base de datos:
mysql community version

Ejercicio nº 2

*. Escenario 1:



*. Escenario 2:



C2 - Automatizacion

Automatizacion de la infraestructura

Habitualmente resulta cansador realizar las mismas tareas rutinarias en el área de sistemas. ¿Por qué mejor no automatizamos la infraestructura? Es decir, que esa tarea la realice un sistema sin errores y más eficientemente. En esta clase vas a conocer cómo implementar la automatización en infraestructura.

[Ver PDF: Automatización de la infraestructura..pdf](#)

Automatización de la infraestructura

Como vimos anteriormente, **la Infraestructura IT es el conjunto de dispositivos y aplicaciones de software necesarios para que cualquier empresa opere.** Esta se compone de elementos como:

- software,
- hardware,
- redes,
- instalaciones
- y todo lo que se requiera para desarrollar, controlar, monitorear y dar soporte a los servicios que ofrece el departamento de IT. Por otro lado, **la automatización consiste en usar la tecnología para realizar tareas casi sin necesidad de las personas.** Se puede implementar en cualquier sector en el que se lleven a cabo tareas repetitivas.

La automatización de la IT —también denominada automatización de la infraestructura— **consiste en el uso de sistemas de software para crear instrucciones y procesos repetibles a fin de reemplazar o reducir la interacción humana con los sistemas de IT.**

¿Por qué automatizar?

Automatizar tareas permite ganar tiempo y maximizar la productividad de nuestra infraestructura IT. En tiempos de **cloud computing**, este es el mensaje que se repite una y otra vez: **cómo hacer más con menos**, cómo conseguir que los profesionales IT de nuestra empresa dediquen más tiempo a generar valor para la compañía y menos a tareas repetitivas que se podrían realizar de forma automática.

Beneficios de automatizar

- Elevar la productividad empresarial.
- Reducir costos operativos.
- Tener una mejor capacidad de respuesta.
- Facilidad de adaptación.
- Disminuir los riesgos de fallas.
- Elevar la seguridad de la información.
- Alojar una mayor cantidad de datos.
- Elevar la competitividad del negocio.

Las 5 tareas más comunes en TI para automatizar



Aprovisionamiento de las áreas de TI.

Automatiza el proceso para habilitar máquinas a través de un levantamiento de servicio para el personal nuevo de nuestra organización.

Gestión de configuración.

Ahorra tiempo y trabajo al automatizar la configuración de las máquinas que se habilitan de acuerdo con los objetivos que tenga el área de TI.



Seguridad y cumplimiento.

Establece acciones de políticas de seguridad y cumplimiento automatizables en la gestión de las máquinas.

Organización de la nube.

Asegura la información y mantiene la disponibilidad de ella a través de la automatización para generar mayor eficiencia.



Implementar aplicaciones.

Algunas aplicaciones requieren instalarse o configurarse, esta tarea puede ser automatizada en las áreas de TI.

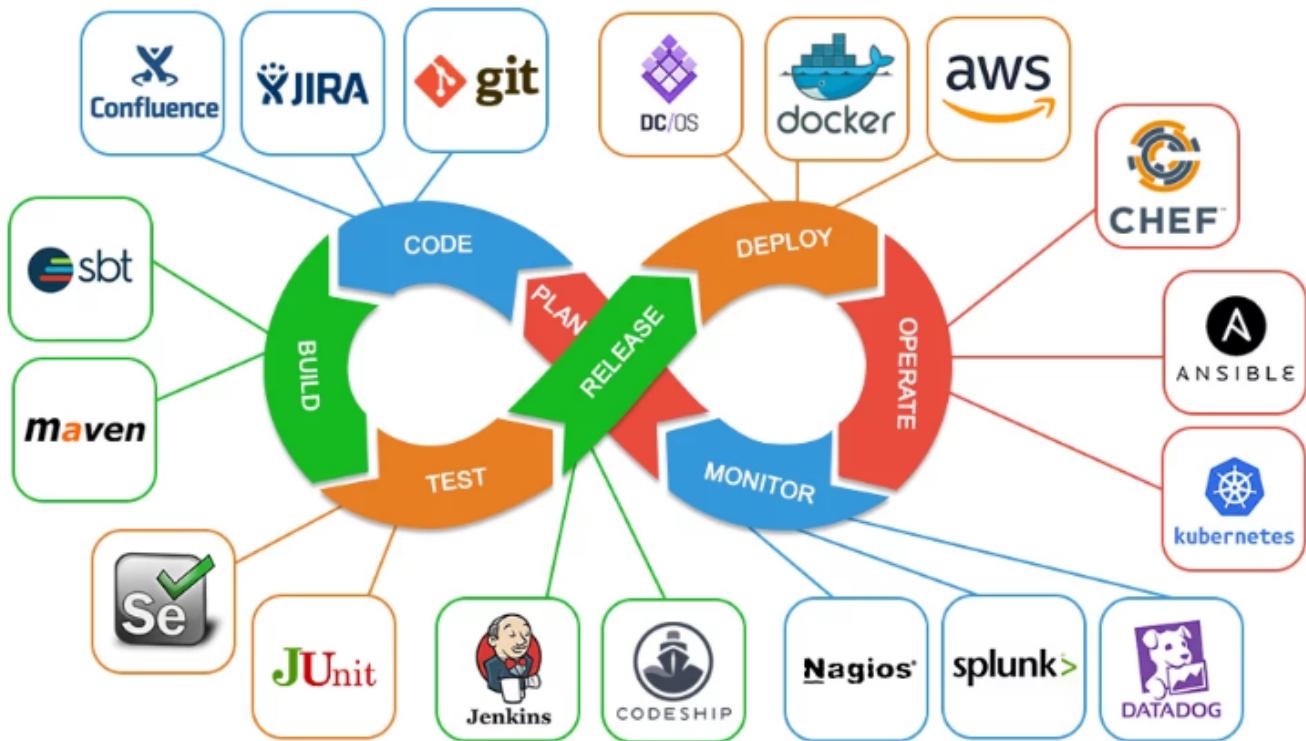
[Ver PDF: Configuración y Mantenimiento del sistema.pdf](#)

Configuración y mantenimiento del sistema

Retomando el concepto que vimos:

"La automatización de la infraestructura consiste en el uso de sistemas de software para crear instrucciones y procesos repetibles a fin de reemplazar o reducir la interacción humana con los sistemas de TI".

Este plantea una visión muy amplia, ya que tiene que abarcar los diversos rubros de las empresas que poseen su área de sistema. Según el tipo de organización en la que trabajamos, es el grado de automatización con el que nos vamos a encontrar en la infraestructura de TI. Uno de los roles del equipo de trabajo del área de sistemas es el **DevOps** engineer, encargado de **automatizar** los procesos del área de **Development** y **Operations**. Su función es hacer más amigable y eficiente la relación entre estos dos equipos de trabajo.



Infraestructura y servicios

Primeramente tenemos que analizar los dispositivos que tenemos en nuestra red de IT. Si contamos con servidores **legacy** o servicios contratados por un **cloud providers**. De las dos opciones anteriores, lo que tenemos que tener en cuenta es el sistema operativo con el que trabajan (MAC, Linux, Windows).

De los cloud providers más conocidos contamos con:

- AWS
- Google Cloud
- Microsoft Azure



Para la elección se evalúan los costos y la oferta de **servicios que disponen**. Esta última es cada vez más robusta y completa (load balancing, backup, clustering, security, etc.).

Manejo del código

Partimos desde donde alojamos nuestro código de trabajo si somos programadores. Por excelencia el gestor de versionado e integrador de código más utilizado es **Git**. Los alojamientos de estos repositorios más conocidos en la nube son GitLab, GitHub, Bitbucket, Gitea, entre otros. En estos se almacena nuestro código de las aplicaciones que pueden estar en lenguajes de programación diversos según lo que necesitemos —por ejemplo: Python, JavaScript, Java, .NET, etc.—. También podemos automatizar el deploy de nuestro código con CI/CD (refiere a las prácticas combinadas de integración continua y entrega continua).

Algunas herramientas para hacerlo son:

- Jenkins
- GitActions
- JetBrains

Contenedores

Los contenedores se están convirtiendo en el modelo de empaquetado de software del producto que desarrollamos. Permiten la virtualización de ambientes de trabajo compatibles transportables, totalmente configurados para que nuestro código funcione en todos los equipos. El más popular es **Docker**.

Cuando manejamos muchos contenedores, tenemos que migrar a un clúster de contenedores, dirigidos por los orquestadores de contenedores. Por ejemplo: Kubernetes o Docker Swarm.

Ambientes de trabajo

Luego, hay que tener en cuenta en qué ambiente está nuestro clúster.

En cada ambiente se trabaja con distintas tecnologías, según el grado de exposición del producto. Automatizando estos procesos vamos a tener más eficiencia, transparencia, facilidad de replicación y recuperación.

Los ambientes más populares son:

- Ansible
- Chef
- Puppet
- Terraform

Monitores de red

Es muy importante tener la supervisión de los dispositivos que hay en nuestra red y de los servicios, y programar alertas en caso de que suceda algún cambio.

Para esto podemos utilizar:

- Nagios,
- Prometheus,
- Icinga2 o
- DataDog.

Lenguajes de scripting

Necesitamos trabajar en estrecha colaboración con los desarrolladores y el administrador del sistema para automatizar tareas de los operadores y desarrolladores (como pueden ser backups, cron jobs, system monitoring).

Según el sistema operativo, podemos usar:

- Bash o

- PowerShell.

Pero también existen lenguajes de scripting independientes del SO, como:

- Python,
- Ruby o
- Go. El más popular es **Python**, ya que posee muchas librerías y es de fácil lectura y aprendizaje.

Conclusión

Estas tecnologías expuestas para automatizar la infraestructura TI no son todas las que están en el mercado, sino una visión general de las que hoy son las más populares. Hay que tener en cuenta que cada día se están desarrollando nuevas herramientas y metodologías de trabajo para cada empresa, como también empresas que brindan el servicio de automatización outsourcing de la organización. **Siempre hay que tomar la decisión basándose en la necesidad, recursos disponibles y experiencia que cuenta la empresa.**

Virtualizacion

¿Cómo viene todo hasta acá? Ahora vamos a refrescar algunos conocimientos que ya vimos en Introducción a la Informática, principalmente la virtualización, dado que es una de las formas de automatización más utilizadas en la infraestructura IT.

Introducción a la virtualización

La **virtualización** permite mejorar la agilidad, la flexibilidad y la escalabilidad de la infraestructura de IT, al mismo tiempo que proporciona un importante ahorro de costos.

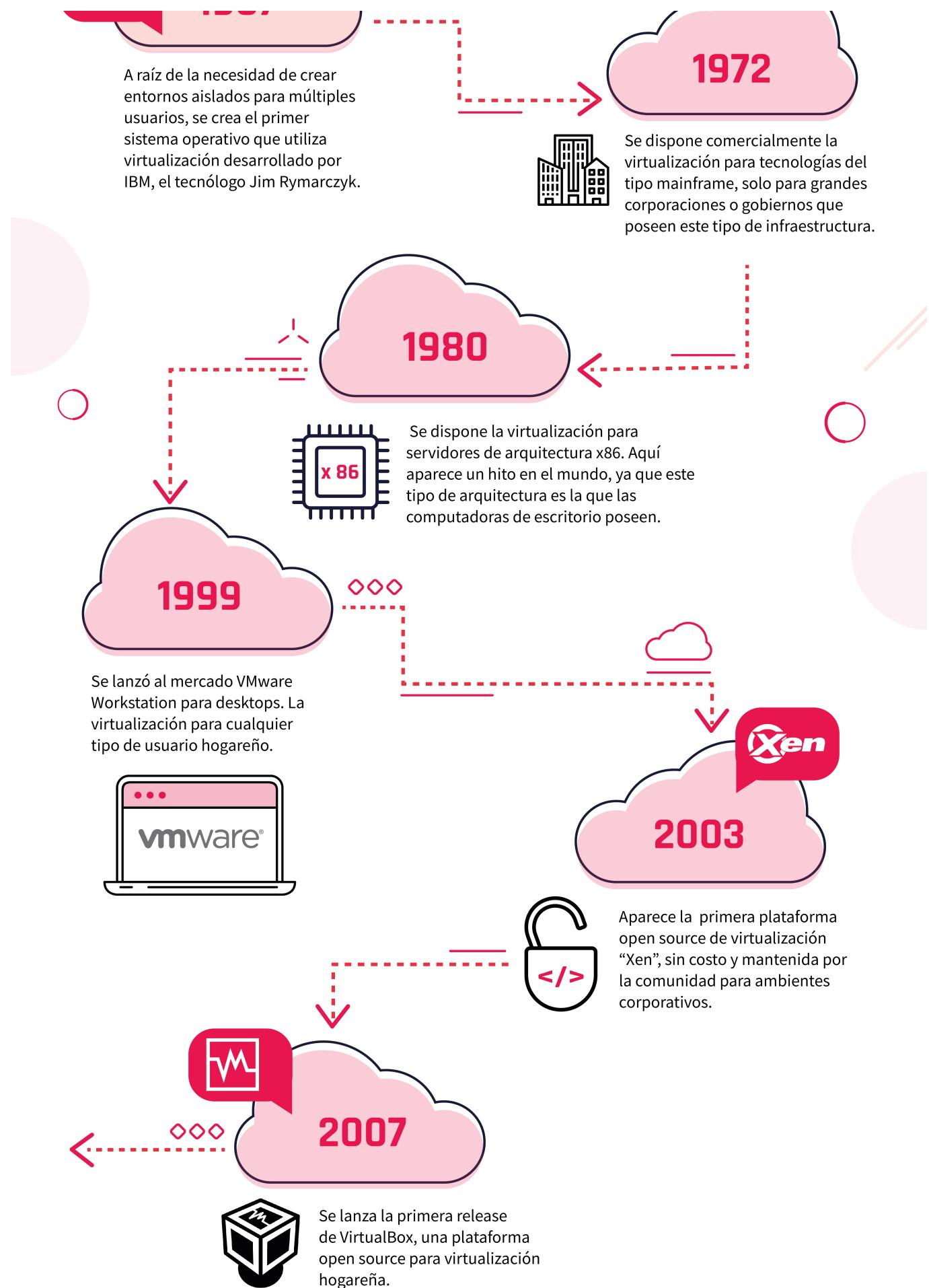
Algunas **ventajas** de la virtualización son:

- Mayor movilidad de las cargas de trabajo.
- Aumento del rendimiento.
- Menos esfuerzo en los upgrades/updates del sistema.
- Mejor disponibilidad de los recursos o la automatización de las operaciones:
- Simplifican la gestión de la infraestructura de IT.
- Permiten reducir los costos de propiedad y operativos.

A continuación, encontrarás la historia de la virtualización y sus componentes.

Historia de la virtualizacion





Componentes de la virtualización

Componentes

Componentes de virtualización

Máquinas virtuales
(ejecución de sistemas operativos)

Máquinas virtuales
(ejecución de sistemas operativos)

Máquinas virtuales
(ejecución de sistemas operativos)

Administrador de máquinas virtuales

Sistemas operativo base

Hardware (servidores físicos)

Maquinas virtuales

En la actualidad se pueden crear sistemas operativos guest del tipo Microsoft y Linux en casi todas sus versiones, se deben tener en cuenta las versiones de cada sistema operativo y la compatibilidad con el sistema host.

Administrador de maquinas virtuales

Desde la herramienta de management se administran todos los recursos físicos y virtuales de los guest que son las instancias virtuales que se crean para usos específicos. Desde el mismo management podremos

establecer clustering con otros virtual machines manager para tener alta disponibilidad y tolerancia a fallos. Además, administra todos los recursos virtuales de nuestras virtual machines.

Sistema operativo base

Es el sistema operativo encargado de administrar los dispositivos físicos (hardware) y proveer una capa de abstracción a los entornos virtuales.

Hardware (Servidores fisicos)

Los microprocesadores, tanto los de Intel como los de AMD, tienen una característica llamada virtualización de CPU. Se aplica a servidores o máquinas de escritorio.

Ver pdf: Instalacion virtual box y vagrant

Ver actividad: crear mv

actividad sincronica

para pasar a root

```
su -
```

- **APT** es un proyecto gigante y su plan original incluia una interfaz gráfica. Está basado en una biblioteca que contiene la aplicación central y apt-get fue la primera interfaz — basada en la línea de órdenes — desarrollada dentro del proyecto. apt es un segundo frontend de linea de comandos proporcionado por APT el cual soluciona algunos errores de diseño de la orden apt-get.
- **Su** El programa su permite usar el intérprete de comandos de otro usuario sin necesidad de cerrar la sesión actual. Comúnmente se usa para obtener permisos de root para operaciones administrativas sin tener que salir y reentrar al sistema.
- El servidor HTTP Apache es un servidor web que ofrece muchas y potentes funciones. Entre ellas se incluyen módulos que cargan de forma dinámica, soporte de medios robusto y una amplia integración con otro software popular.
- Me resolto familiar, como un escritorio remoto

Notas - Clase 2: Virtualizacion

Uso de virtualizacion para testeo y aprovechar recursos de hardware. Saber el comportamiento de una aplicación en un sistema operativo. Al hacer un servidor quizás sirva virtualizarlo, aprovechando los recursos de hardware de esa forma.

recomendado estar con red cableada para la instalación de **Debian** la versión más común es AMD64
servidores Legacy se refiere a los servidores físicos.

Instalacion VM Debian, Apache2, OpenSSH, puTTy

ver instalacion vm Debian en min 09:00

Ver PDF: Crear una VM

Link video: habilitar virtualizacion [link](#)

instalamos la vm en oracle vm, para administrar un servidor sin un entorno grafico

GRUB es un gestor de arranque que se instala en el disco, antes de la particion donde teniendo ese GRUB podemos tener varias instalaciones en el mismo disco de distintos sistemas operativos

vamos a instalar un servidor web y lo vamos a consultar desde nuestra maquina host. Vamos a ver la pagina web que esta alojada en este servidor que estamos haciendo

Terminada la instalacion y configuracion de nuestra maquina cirtual Debian:

```
login: nombre  
clave: clave
```

Procedemos a instalar el servidor web apache, el mas conocido en linux, mas antiguo y open source

cambiamos el login a root para realizar cambios de administrador

```
su root
```

instalamos apache2

```
apt-get install apache2
```

pide un S para continuar y bajara unos archivos e instalara el servidor web

verificamos que el servidor este andando en la computadora con:

```
ip address
```

ip address muestra los dispositivos de red que tenemos en nuestra maquina virtual

```
root@debiancliente:/home/liliana# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group defau
000
    link/ether 08:00:27:89:d7:60 brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic enp0s3
        valid_lft 85783sec preferred_lft 85783sec
    inet6 fe80::a00:27ff:fe89:d760/64 scope link
        valid_lft forever preferred_lft forever
root@debiancliente:/home/liliana#
```

muestra 2 dispositivos, el primero es **lo** hace referencia al **local host** siempre es **127.0.0.1**, no la tenemos que usar ya que hace referencia a su misma placa de red.

la que tenemos que ver es la 2 **enp0s3: 192.168.1.25** que la direccion que tiene esta en el mismo rango que la direccion ip de la maquina host.

Verificamos esto fuera de nuestra maquina virtual, en nuestra maquina host



Ahora instalaremos un servidor SSH, desde el root, este servidor nos da una puerta de entrada a nuestra maquina virtual desde afuera, desde este protocolo ssh. (Como un team viewer para terminales, con esto se hace mantenimiento de servidores)

```
apt-get install openssh-server
```

cuando ponemos esto crea ciertas dependencias y se instala el servidor ssh, al mismo tiempo usaremos la app **puTTY** en nuestro host.

en host name ponemos la ip que ip address **192.168.1.25** nos sale un warning y le ponemos que si.

y ya nos conecta, usamos el login del usuario normal no el root, ya que ssh no deja root.

```

liliana@debiancliente: ~
login as: liliana
liliana@192.168.1.25's password:
Linux debiancliente 5.10.0-8-amd64 #1 SMP Debian 5.10.46-4 (2021-08-03) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Sep 19 18:34:22 2021
liliana@debiancliente:~$ 
```

esto nos sirve para usar vagrant y nos acerca a la automatizacion.

PutTY is a free implementation of Telnet and SSH for Windows and Unix platforms, along with an xterm terminal emulator.

Notas - Clase 3: Repaso

Vagrant: Automatizacion box Debian

Vamos a usar vagrant para automatizar la virtualizacion. Con un script con vagrant lo haremos mucho mas rapido

paso a paso:

- Creamos una carpeta llamada ejVagrant y accedemos desde PowerShell

```

PS C:\Users\Leandro> cd C:\Users\Leandro\Desktop\Li\courses\CTD\Bimestre_II\infraestructura_i\C3-
Repasso
PS C:\Users\Leandro\Desktop\Li\courses\CTD\Bimestre_II\infraestructura_i\C3-Repasso> ls

Directory: C:\Users\Leandro\Desktop\Li\courses\CTD\Bimestre_II\infraestructura_i\C3-Repasso

Mode                LastWriteTime         Length Name
----                -----          ---- -
d-----        9/16/2021    1:47 PM            ejerciciosVagrant
d-----        9/19/2021    7:56 PM            ejVagrant
-a---        9/16/2021    2:38 PM           472 vagrantfile
-a---        9/16/2021    2:37 PM       299064 VM_con_Vagrant__Virtualbox.pdf

PS C:\Users\Leandro\Desktop\Li\courses\CTD\Bimestre_II\infraestructura_i\C3-Repasso> cd ejVagrant
PS C:\Users\Leandro\Desktop\Li\courses\CTD\Bimestre_II\infraestructura_i\C3-Repasso\ejVagrant>
```

- con vagrant instalado, descargaremos la box de Debian, una box es una maquina virtual preconfigurada, vemos las diferentes boxes en la vagrant cloud:

<app.vagrantup.com/boxes/search>

nosotros usaremos:

una box de Debian y usaremos virtualbox como hypervisor, se llama `debian/buster`

ejecutamos el comando:

```
vagrant box add debian/buster64
```

esto demora un poco, luego seleccionamos la opcion 2: virtualbox

el **vagranfile** que tiene el siguiente codigo con extension `all types`, configura la maquina virtual y le pone nombre `network server` y que configure una `public network`

```
# -*- mode: ruby -*-
# vi: set ft=ruby :
# All Vagrant configuration is done below. The "2" in Vagrant.configure
# configures the configuration version (we support older styles for
# backwards compatibility). Please don't change it unless you know what
# you're doing.
Vagrant.configure("2") do |config|
  config.vm.define "server" do |server|
    config.vm.box = "debian/buster64"
    server.vm.hostname = "server"
    server.vm.network "public_network"
  end
end
```

validamos el codigo:

```
vagrant validate
```

si hacemos

```
vagrant up
```

y hay un error, tambien lo va a validar con `vagrant up` corre la maquina virtual

Todo el proceso de la clase 2 de instalar y configurar el sistema operativo vagrant lo va a automatizar; lo ahorraremos con la box.

Cuando termine de instalar con el comando verificamos que esta corriendo.

```
vagrant status
```

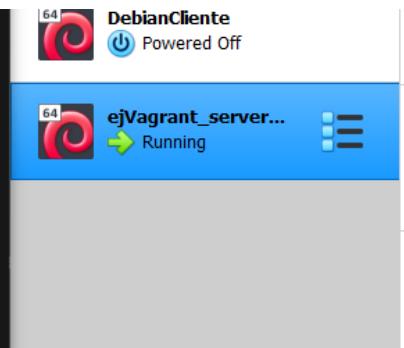
Si dice running y abrimos el virtualbox vemos que esta corriendo

```
PS C:\Users\Leandro\Desktop\Li\courses\CTD\Bimestre_II\infraestructura_i\C3-Repaso\ejVagrant> vagrant status
Current machine states:

server           running (virtualbox)

The VM is running. To stop this VM, you can run `vagrant halt` to
shut it down forcefully, or you can run `vagrant suspend` to simply
suspend the virtual machine. In either case, to restart it again,
simply run `vagrant up`.

PS C:\Users\Leandro\Desktop\Li\courses\CTD\Bimestre_II\infraestructura_i\C3-Repaso\ejVagrant>
```



ahora vamos a instalar un servidor web, vamos a tener que loggearnos a esta maquina virtual

Como vagrant ya estala un ssh usamos el siguiente codigo y esto evita el uso de puTTy

```
vagrant ssh server
```

el comando anterior nos logea automaticamente a la maquina virtual

ahora vamos a instalar un servidor web: apache, para eso necesitamos los permisos de root, vagrant es el password por defecto de root.

```
su root
Password: vagrant
# Por defecto el password de root es vagrant
```

ahora que estamos como root instalaremos el servidor web apache2

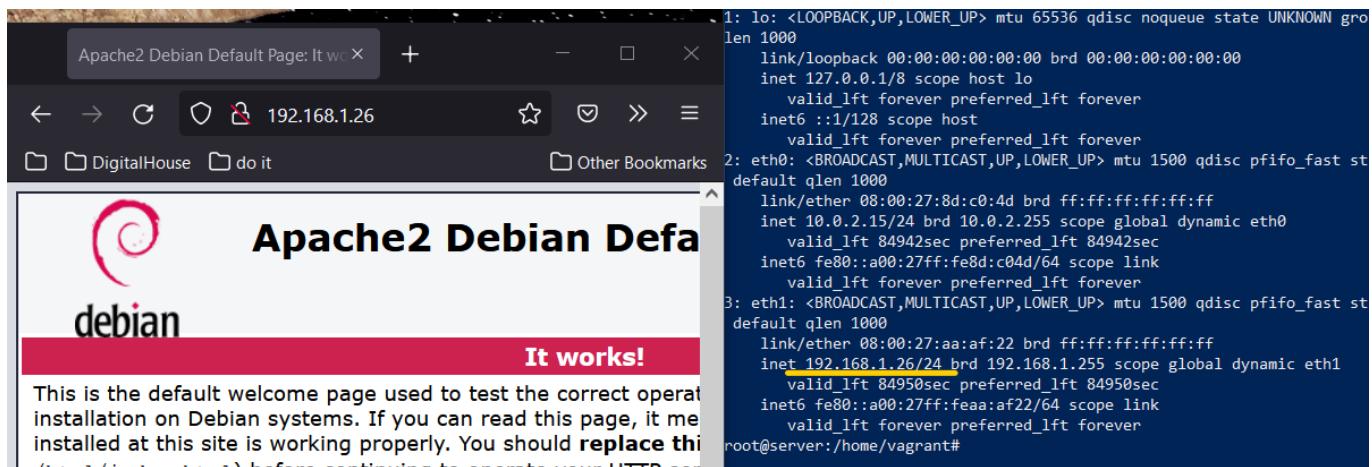
```
apt-get install apache2
```

pregunta, le decimos Y, lo va a bajar, configurar y lo va a instalar.

para ver si esta funcionando nos fijamos la ip

```
ip address
```

Verificamos que funcione en el navegador que corre nuestro servidor web



Resumen Esto automatiza la tarea de tener un servidor web usando los recursos de hardware de nuestra computadora, antes lo usamos de forma manual en la clase 2 y en la clase 3 se automatiza el proceso.

Podemos cambiar las configuraciones cuando esta instalado en nuestra computadora, yendo a la documentacion de en este caso [Debian](#)

Las boxes se guardan en:

```
C:\usuarios>user> .vagrant.d>boxes
```

Automatizaremos mas el proceso anterior, agregando por defecto al script un servidor web primero saldremos del root y de la maquina virtual.

```

root@server:/home/vagrant# exit
exit
vagrant@server:~$ exit
logout
Connection to 127.0.0.1 closed.
PS C:\Users\Leandro\Desktop\Li\courses\CTD\Bimestre_II\infraestructura_i\C
3-Repaso\ejVagrant>

```

para parar nuestra maquina virtual:

```
vagrant halt
```

Destruir la maquina de vagrant con:

```
vagrant destroy -f
```

Automatizar: modificando file de vagrant

Agregamos las intrucciones al script para que se instale automaticamente el [Apache2](#)

```
Vagrant.configure("2") do |config|
  config.vm.define "server" do |server|
    config.vm.box = "debian/buster64"
    server.vm.hostname = "server"
    server.vm.network "public_network"
    #Esto es lo nuevo que se agrega al file
    #Le dice al server que se aprovisione desde la linea de comandos
    server.vm.provision "shell", inline: <<-SHELL
      #Y ejecute estos comandos:
      apt-get update
      apt-get install -y apache2
      #Dice que termine el shell
      SHELL
    end
  end
```

Primero lo validamos:

```
vagrant validate
```

si esta ok

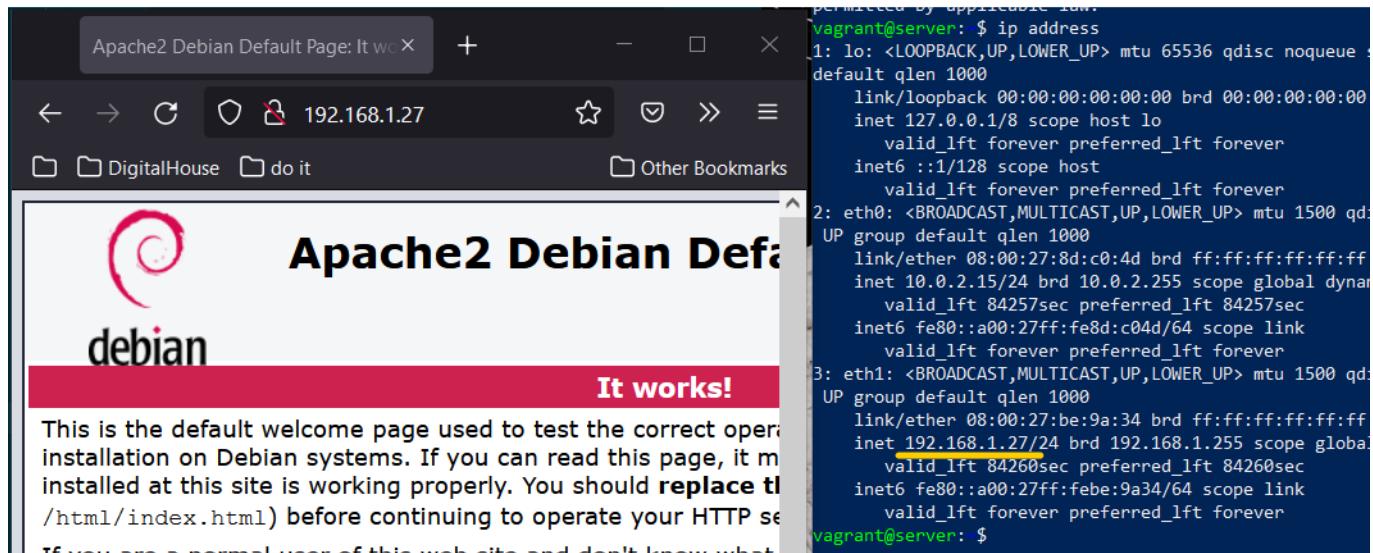
```
vagrant up
```

Nos logueamos para ver la ip, aunque tambien podemos configurarla

```
vagrant ssh server
```

y luego, no es necesario entrar como root, pedimos el ip y lo verificamos

```
ip address
```



Cerramos la vm

```
#loguados en el servidor con vagrant ssh server
exit
```

y eliminamos la VM con:

```
vagrant destroy -f
```

Cambiar html base en el script

Copiar el [HTML](#), ponerlo en la carpeta vagranfile para copiar esta pagina en donde iria la pagina por defecto de Apache.

```
Vagrant.configure("2") do |config|
  config.vm.define "server" do |server|
    config.vm.box = "debian/buster64"
    server.vm.hostname = "server"
    server.vm.network "public_network"
    server.vm.provision "shell", inline: <<-SHELL
      apt-get update
      apt-get install -y apache2
    SHELL
    #Esto es lo nuevo que se agrega al file
    server.vm.provision "file", source: "index.html", destination: "index.html"
  end
end
```

```
server.vm.provision "shell", inline: "mv index.html /var/www/html/index.html"
end
end
```

Esto movera el archivo HTML a ese lugar, tiene que existir ese HTML

Creamos una pagina html en la carpeta del vagrant file

```
#paso 1: Correr vagrant
vagrant up
#paso 2: Elegir la red
1
#paso 3: Nos logueamos en la maquina virtual
vagrant ssh server
#paso 4: Consulto ip
ip address
```

Liliana Ospina

```
vagrant@server: ~
PS C:\Users\Leandro\Desktop\Li\courses\CTD\Bimestre_II\infraestructura_i\C3-nuevo> vagrant ssh server
Linux server 4.19.0-17-amd64 #1 SMP Debian 4.19.194-3 (2021-07-18) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
vagrant@server:~$ ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:8d:c0:4d brd ff:ff:ff:ff:ff:ff
    inet 10.0.2.15/24 brd 10.0.2.255 scope global dynamic eth0
        valid_lft 86176sec preferred_lft 86176sec
    inet6 fe80::a00:27ff:fe8d:c04d/64 scope link
        valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 08:00:27:fb:3a:82 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.30/24 brd 192.168.1.255 scope global dynamic eth1
        valid_lft 86185sec preferred_lft 86185sec
    inet6 fe80::a00:27ff:fefb:3a82/64 scope link
        valid_lft forever preferred_lft forever
vagrant@server:~$
```

(Se ve en otra pc en mi misma red, no se si afuera) (NO MATE ESTA ULTIMA MAQUINA VIRTUAL) (UTIMO COMANDO FUE `vagrant halt`)

PARA PRENDERLA `vagrant up`

C4 - Shell Scripting - Parte I

Introducción a la terminal de Linux

¿Cómo viene todo hasta acá? Ojalá hayas podido recuperar energías porque comienza el momento de trabajar sobre nuevas definiciones.

En esta clase vamos a avanzar respondiendo algunas preguntas:

- ¿Qué es una shell CLI o intérprete de comandos?
- ¿Cuáles son los tipos de shell?
- ¿Cómo iniciamos una consola en Linux?
- ¿Cuáles son los privilegios del superusuario root?

ver PDF: Introducción a la Terminal. Diferentes formas de ejecución.pdf

La consola de Linux

La interfaz de línea de comandos, o CLI —por sus siglas en inglés command-line interface—, es un método de comunicación entre usuario y máquina que acepta instrucciones del usuario a través de líneas de texto (siguiendo unas determinadas reglas de sintaxis que puedan ser interpretadas por el sistema operativo). La herramienta que posibilita la función de interfaz de usuario se la denomina shell. Aplicado en el ámbito de la interfaz de línea de comandos, estaríamos hablando de una shell CLI o intérprete de comandos. Diferentes

Tipos de shell

En Linux tenemos una multitud de shells o intérpretes diferentes. El más conocido de todos probablemente es Bash, debido a que es el que suele venir por defecto en la gran mayoría de distribuciones GNU/Linux, pero también destacan otros como Bourne Shell (sh), Korn Shell (ksh) o C Shell (csh), los cuales vamos a conocer.

Bourne Shell

Lleva el nombre de su creador en los Laboratorios Bell, Steve Bourne. Fue la primera shell utilizada para el sistema operativo Unix y ha superado en gran parte la funcionalidad de muchas de las shells más recientes. Todas las versiones de Linux Unix permiten a los usuarios cambiar a la original Bourne Shell, conocida simplemente como "sh", si así lo desean. Sin embargo, hay que tener en cuenta que al hacerlo, se renuncia a funcionalidades como el completado de nombres de archivo y el historial de comandos que los depósitos posteriores han añadido.

C/TC Shell

El C Shell fue desarrollado posteriormente al Bourne Shell y está pensado en facilitar el control del sistema al programador en lenguaje C. La razón de esto es que su sintaxis, como vamos a apreciar, es muy similar a la de este lenguaje. Conocido popularmente también como csh, está presente en otros SO, por ejemplo, en Mac OS. Posee una evolución, conocida como tcsh que incorpora funcionalidades avanzadas y mayores atajos de teclado.

Korn Shell

Esta también fue escrita por un programador en los Laboratorios Bell, David Korn. Intenta combinar las características de la C Shell, TC Shell y Bourne Shell en un solo paquete. También incluye la capacidad para crear nuevos comandos de shell para los desarrolladores cuando surja la necesidad.

Posee funciones avanzadas para manejar archivos de comandos que la colocan a la par de lenguajes de programación especializados, como AWK y Perl.

Bourne-Again Shell (BASH)

La Bourne-Again Shell es una versión actualizada de la Bourne Shell original. Es una shell utilizada ampliamente en la comunidad de código abierto. Su sintaxis es similar a la utilizada por la Bourne Shell, incorporando funcionalidades más avanzadas que se encuentran en las shells C, TC y Korn. Entre las funcionalidades adicionales que carecía Bourne, está la capacidad para completar nombres de archivos pulsando la tecla TAB, la capacidad de recordar un historial de comandos recientes y la capacidad de ejecutar múltiples programas en segundo plano a la vez.

Ejecución de la consola

Consola de Linux: Ejecución en inicio

Si bien cada distribución de Linux tiene su manera particular de acceder a la consola, cuando el SO se inicia en los niveles 1, 2, 3 y 4 nos llevará por defecto a la consola.

Consola de Linux: Ejecución desde GUI

Si en cambio nuestro SO inicia en nivel 5 (con GUI), para poder utilizar la terminal tenemos diferentes opciones. Estas varían de acuerdo a la distribución instalada. En el caso de Ubuntu, tenemos dos opciones:

- La primera de ellas es lanzando un TTY, o espacio de trabajo sin entorno gráfico. Podemos ejecutar 7 terminales al mismo tiempo de esta forma. De la 1 a la 6, ninguna tiene interfaz gráfica. Para cambiar de TTY en Linux debemos usar el atajo de teclado Control+Alt más la tecla —de F1 al F7— del TTY que queramos ejecutar.

`ctrl + alt + f1`

- La segunda opción es encontrar una app dedicada que se ejecuta en una ventana, dentro del panel de aplicaciones de nuestra distro. En el caso de Ubuntu, por ejemplo, podemos encontrar esta terminal dentro del cajón de programas del entorno gráfico GNOME.

Los privilegios del superusuario root

Por lo general, los sistemas operativos contemplan el uso de solo un usuario, el cual tiene permisos de administrador. En Linux las cosas se manejan de una forma particular, se separa la cuenta de usuario común de la de superusuario y es eso lo que conocemos como root. Esta cuenta posee todos los privilegios y permisos para realizar acciones sobre el sistema.

Para la ejecución de algunos comandos debemos ingresar dicho acceso (clave de root). Sin embargo, se debe tener un conocimiento sobre las acciones que se realizan, ya que una acción realizada de manera errónea podría ocasionar daños importantes en el sistema. El uso de instrucciones con privilegios de superusuario pueden ser sumamente útiles, pero totalmente devastadoras si desconocemos las consecuencias de su uso en el sistema. Veamos el método para elevar nuestros privilegios.

Elevando privilegio

Suponiendo que iniciamos sesión como un usuario "común", denominado "edorio" y queremos reiniciar un servicio (cron), vamos a obtener lo siguiente:

```
edorio@DESKTOP-W10:~$ service cron start
* Starting periodic command scheduler cron
cron: can't open or create /var/run/crond.pid: Permission denied
[fail]
edorio@DESKTOP-W10:~$
```

Para evitar el error, debemos usar el comando sudo, previo al comando que queremos ejecutar. Nos pedirá la contraseña de root y se ejecutará como tal de manera satisfactoria

```
edorio@DESKTOP-W10:~$ sudo service cron start
[sudo] password for edorio:
 * Starting periodic command scheduler cron
[ OK ]
edorio@DESKTOP-W10:~
```

Comandos más utilizados en la terminal de Linux

A continuación vas a encontrar respuestas a los siguientes interrogantes:

- ¿Cómo navegar por el sistema de archivos mediante la consola?
- ¿Cómo manejar los archivos desde la consola?
- ¿Cómo obtener información desde un web service?

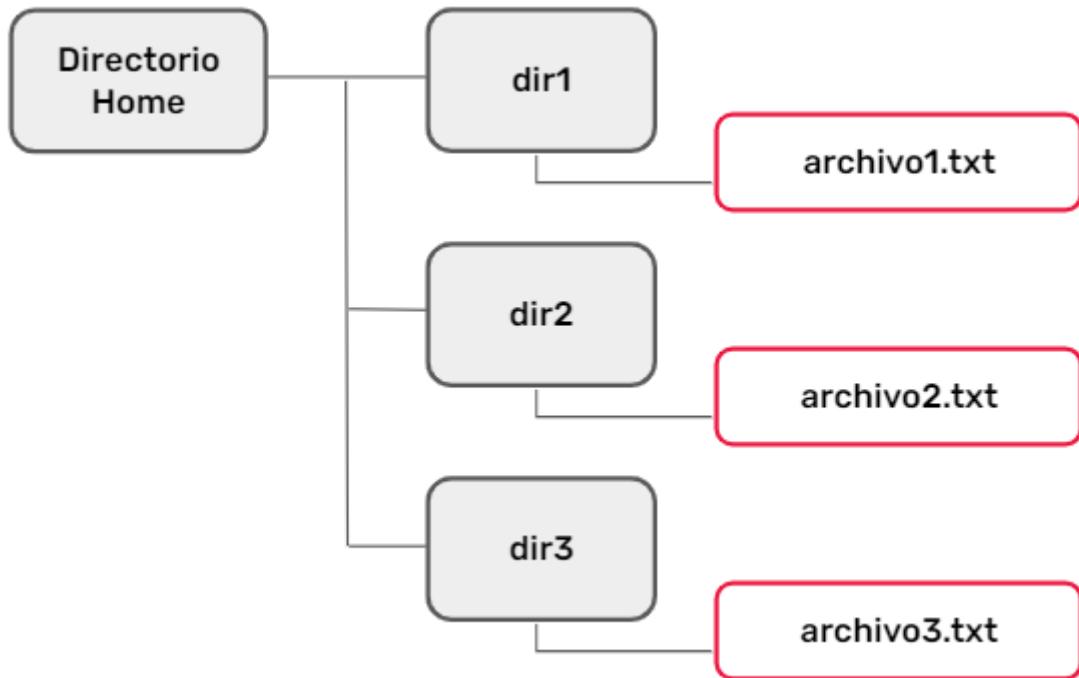
[Ver PDF: Comandos Utiles - parte 1.pdf](#)

Consolidando nuestro ambiente

Para poder seguir correctamente los ejemplos posteriores, es deseable que en tu ambiente (máquina virtual o WSL) tengas replicada la siguiente estructura de carpetas y archivos.

Para ello debemos ejecutar los siguientes comandos, en el orden dado (solo el texto que está luego del prompt o sea, luego del "\$", en color blanco):

```
edorio@DESKTOP-W10:~$ mkdir dir1 dir2 dir3
edorio@DESKTOP-W10:~$ touch dir1/archivo1.txt dir2/archivo2.txt dir3/archivo3.txt
```



Verificando nuestro ambiente

Listamos los directorios con la instrucción `ls -R`. Deberíamos obtener lo siguiente:

```
edorio@DESKTOP-W10:~$ ls -R
.:
dir1  dir2  dir3
./dir1:
archivo1.txt
./dir2:
archivo2.txt
./dir3:
archivo3.txt
```

Comandos para el manejo de archivos

ls

Con el comando `ls` podrás listar los diferentes archivos y directorios de la carpeta de trabajo en la que te encuentres. El comando acepta multitud de opciones, algunas de las cuales veremos a continuación. A continuación, podemos observar el uso más simple del comando `ls`. Si no le indicamos ninguna opción, enumerará todos los archivos y directorios que se encuentran en la carpeta de trabajo actual, sin tener en cuenta archivos ocultos.

```
edorio@DESKTOP-W10:~$ ls
dir1  dir2  dir
```

ls -a

Con esta opción, el comando te mostrará —en forma de lista— todo el contenido que se encuentre dentro del directorio de trabajo, incluyendo archivos y carpetas ocultos. Dependiendo del shell, algunos tipos de archivos se mostrarán con colores diferentes

```
edorio@DESKTOP-W10:~$ ls -a
.           .aws          .bash_logout  .config      .landscape
.profile    .vagrant.d   dir3 ..        .azure
.bashrc     .docker       .local        .ssh        dir1  .ansible
.bash_history .cache      .fastlane   .motd_shown
.sudo_as_admin_successful  dir2
```

ls -l

Esta opción es similar a la anterior, pero muestra el contenido en forma de lista e incluye información referente a cada elemento. Es de las más utilizadas, siendo especialmente útil a la hora de conocer el propietario y los permisos de cada fichero

```
edorio@DESKTOP-W10:~$ ls -l
total 12
drwxr-xr-x 2 edorio edorio 4096 May 21 01:59 dir1
drwxr-xr-x 2 edorio edorio 4096 May 21 01:59 dir2
drwxr-xr-x 2 edorio edorio 4096 May 21 01:59 dir3
```

mkdir

Te permitirá crear un directorio con el nombre y la ruta que especifiques. Si no le indicás ninguna ruta, por defecto, te creará la carpeta dentro del directorio de trabajo en el que te encuentres

```
edorio@DESKTOP-W10:~$ mkdir dir4
```

Caso contrario, le podés indicar que cree un directorio con un path definido dentro de dir1.

```
edorio@DESKTOP-W10:~$ mkdir dir1/subdir1
```

rmdir

Te permite eliminar el directorio que le especifiques. Un detalle importante es que para poder utilizar este comando, el directorio a borrar debe estar vacío

```
edorio@DESKTOP-W10:~$ rmdir dir4
```

El de arriba es el uso más simple del comando, sin indicar ruta.

Podemos también borrar un directorio con un path definido.

```
edorio@DESKTOP-W10:~$ rmdir dir1/subdir1
```

rm

Este comando permite eliminar archivos sueltos y directorios que no se encuentren vacíos.

```
edorio@DESKTOP-W10:~$ rm dir1/archivo1.txt
```

El de arriba es el uso más simple del comando, sin indicar ruta. Eliminamos 1 archivo específico dentro de dir1

```
edorio@DESKTOP-W10:~$ rm -r dir2
```

Con el modificador -r eliminamos el directorio dir2 y, recursivamente, todo su contenido. Es un comando a utilizar con mucha precaución

cp

Usando este comando serás capaz de copiar archivos y directorios. Así como ubicarlos en otras rutas, definiendo origen primero y luego destino.

```
edorio@DESKTOP-W10:~$ cp dir3/archivo3.txt dir1/archivo1.txt
```

El de arriba es el uso más simple del comando, sin indicar ruta. Copiamos en este caso el archivo3.txt, hacia dir1 y lo nombramos archivo1.txt. En este caso, con el modificador -r copiamos el directorio dir3 en uno llamado dir2, que el mismo comando creo.

```
edorio@DESKTOP-W10:~$ cp -r dir3 dir2
```

mv

Este comando te servirá para mover archivos desde la consola. La sintaxis es muy sencilla, solamente deberás especificar la ubicación de inicio —incluyendo el nombre del archivo— y la ubicación de destino.

```
edorio@DESKTOP-W10:~$ mv dir1/archivo1.txt dir3/archivo1.txt
```

Movimos un archivo de dir1 a dir3, conservando su nombre original. En el siguiente caso usamos el comando mv para renombrar un archivo, ya que las rutas definidas son las mismas.

```
edorio@DESKTOP-W10:~$ mv dir3/archivo1.txt dir3/archivo3bis.txt
```

Comandos para leer archivos de texto

cat

Este es uno de los comandos más utilizados cuando se trata de manejar archivos de texto (en formato .txt) desde la terminal. Entre sus múltiples opciones, está la posibilidad de crear un archivo e imprimir por pantalla su contenido.

```
edorio@DESKTOP-W10:~$ cat >dir1/archivo1.txt
```

Esto nos abrirá el archivo1.txt, permitiendo editararlo. Con la combinación CTRL+D terminaremos la edición y se guardará el contenido.

```
edorio@DESKTOP-W10:~$ cat dir1/archivo1.txt
Hola Digital House, esto es CAT
```

Invocando el comando sin el símbolo ">", nos mostrará por pantalla el contenido del mismo. Se puede usar con el modificador -n, para numerar las líneas y con el -b, con el propósito de no mostrar las líneas en blanco.

more

Este es otro comando útil para imprimir por pantalla el contenido de un archivo de texto. Esencialmente es igual que el comando cat, con la diferencia que este comando pagina el contenido, por lo que es más adecuado para leer archivos largos.

```
edorio@DESKTOP-W10:~$ more /var/log/dpkg.log
```

Nos paginará el archivo en cuestión, de tal manera que en sus últimas líneas lo veremos así:

```
2021-02-19 23:56:28 remove linux-headers-5.4.0-65:all 5.4.0-65.73 <none>
2021-02-19 23:56:28 status half-configured linux-headers-5.4.0-65:all 5.4.0-65.73
2021-02-19 23:56:28 status half-installed linux-headers-5.4.0-65:all 5.4.0-65.73
--More-- (5%)
```

nano

Nano es un editor de textos para la terminal, que más que para leer archivos sirve para modificarlos y editarlos. Aunque para esta guía también nos vale perfectamente para abrir el archivo y visualizar su

contenido desde la línea de comandos.

```
edorio@DESKTOP-W10:~$ nano dir1/archivo1.txt
```

Nos mostrará:



GNU nano 4.8 dir1/archivo1.txt
Hola Digital House, esto es CAT

Una vez abierto, en la parte inferior se visualizará las diferentes combinaciones de teclas que necesitarás a la hora de trabajar con archivos.

En la parte inferior se muestran las diferentes combinaciones de teclas que se necesitarán a la hora de trabajar con archivos:

- **CTRL+R**: combinación para indicarle un archivo de texto a Nano para que lo abra y muestre su contenido por la consola.
- **CTRL+V**: estando dentro de Nano y con el archivo abierto en la consola, esta combinación sirve para avanzar a la página siguiente.
- **CTRL+Y**: sirve para retroceder a la página anterior.
- O **CTRL+W**: sirve para introducir un carácter o grupo de caracteres y buscar en el texto cualquier letra o palabra que coincida con el parámetro de búsqueda.
- **CTRL+X**: para cerrar el archivo una vez que lo hayas terminado de visualizar en la consola. Eso cerrará el editor de texto Nano y volverá a aparecer el prompt de Bash por consola.

grep

Este comando, perteneciente a la familia Unix, es una de las herramientas más versátiles y útiles disponibles. Se encarga de buscar un patrón que definamos en un archivo de texto. Su primer parámetro es la cadena de texto a buscar, luego el o los archivos (acepta comodines como *, pudiendo con el modificador -r recorrer recursivamente) que vamos a buscar.

```
edorio@DESKTOP-W10:~$ grep "Digital House" * -r
```

En este caso, buscamos la cadena "Digital House" en todos los archivos, de manera recursiva, la ejecución nos devolvió lo siguiente:

```
edorio@AR-CARRERA-09:~$ grep "Digital House" * -r  
dir1/archivo1.txt:Hola Digital House, esto es CAT  
edorio@AR-CARRERA-09:~$
```

tee

Lee una entrada estándar y la escribe en la salida estándar y en uno o más archivos. De forma normal, en la redirección de salida, las líneas del comando se escriben en un archivo, pero si queremos ver dicha salida al

mismo tiempo, no podemos. ¡Usando el comando tee sí es posible lograrlo!

```
edorio@DESKTOP-W10:~$ ls -l | tee listado.txt
```

En este caso, además de mostrarnos el directorio, el mismo será guardado en un archivo

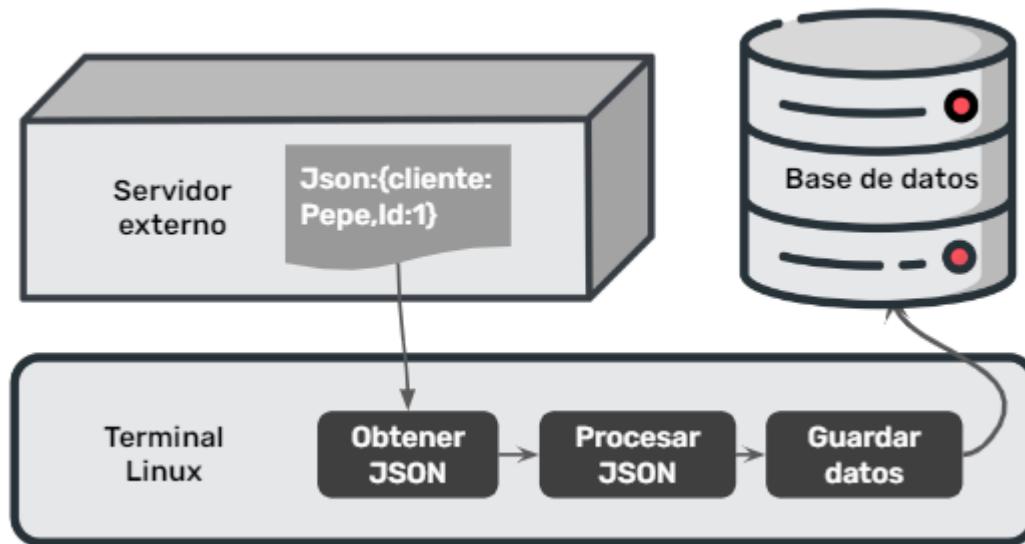
```
edorio@DESKTOP-W10:~$ ls -l | tee -a listado.txt
```

Utilizando el modificador -a, se agregará el contenido al archivo, sin pisar lo anterior.

[Ver PDF: Comandos Utiles - parte 2.pdf](#)

Obtener datos desde un web service

La terminal de Linux tiene tanta versatilidad y potencia que nos permite vincularla con un web service, obtener datos de allí y procesarlos con propósitos tales como agregarlo a archivos en nuestro servidor, modificarlos y republicarlos. Las opciones son muy variadas, por ejemplo obtener un JSON desde una URL externa, procesar su contenido, obtener el o los atributos que nos interesen y —en función de ello— crear nuevos archivos, insertarlos en una base de datos.



Comando cURL

Aspectos técnicos

Es un comando disponible en la mayoría de los sistemas basados en Unix. Es una abreviatura de "Client URL". Los comandos de cURL están diseñados para funcionar como una forma de verificar la conectividad a las URL y como una gran herramienta para transferir datos. El comando tiene una amplia compatibilidad con los protocolos más usados

FTP**HTTP****POP / SMTP / IMAP****SCP**

Sintaxis básica

El uso más simple de cURL es mostrar el contenido de una página. El siguiente ejemplo mostrará la página de inicio de digitalhouse.com:

```
edorio@DESKTOP-W10:~$ curl https://www.digitalhouse.com
```

Como vemos, no es muy útil esto, ya que es difícil llegar a información que nos pueda ser de utilidad visualizándola solamente en pantalla. Pero si usamos el modificador `-o`, podremos escribir ese contenido HTML de la página de inicio en un archivo en nuestro equipo:

```
edorio@DESKTOP-W10:~$ curl https://www.digitalhouse.com -o mipagina.html
```

Esto guardará todo el HTML en el archivo `mipagina.html`.

Descargas

El uso de este modificador puede extenderse a procesar descargas:

```
edorio@DESKTOP-W10:~$ curl https://ubuntu.zero.com.ar/ubuntu-releases/20.04/ubuntu-20.04.2.0-desktop-amd64.iso -o ubuntu.iso
```

Descarga la ISO de la URL de referencia y la nombrará **ubuntu.iso**

```
edorio@DESKTOP-W10:~$ curl https://ubuntu.zero.com.ar/ubuntu-releases/20.04/ubuntu-20.04.2.0-desktop-amd64.iso -O -C 0
```

En este caso, no renombramos el archivo de destino (con el modificador -O). Además, permitimos la continuidad de la descarga con el modificador -C.

Encabezados y verificaciones

El modificador -v nos permite verificar la conectividad hacia un sitio remoto

```
edorio@DESKTOP-W10:~$ curl https://www.digitalhouse.com -v
```

Esto nos brindará, además del contenido, datos como la IP de destino, protocolos de seguridad y certificados utilizados.

```
edorio@DESKTOP-W10:~$ curl https://www.digitalhouse.com -I
```

El modificador -I nos muestra todos los encabezados de la solicitud, tales como ruta por defecto, publicador web, entre otros.

Contenido JSON

Viendo todas las opciones brindadas, nos podemos imaginar lo útil de este comando con el propósito de obtener el contenido en formato JSON desde un endpoint que lo entregue en dicho formato. Por ejemplo, la API de OpenStreetMap, la cual nos devuelve una dirección pasándole las coordenadas, con la siguiente URL: <https://nominatim.openstreetmap.org/reverse.php?lat=-34.60378&lon=-58.38161&zoom=18&format=jsonv2>

```
edorio@DESKTOP-W10:~$ curl "https://nominatim.openstreetmap.org/reverse.php?lat=-34.60378&lon=-58.38161&zoom=18&format=jsonv2" -o resultado.json
```

Allí estamos guardando en el archivo resultado.json lo obtenido en el web service. Notemos el detalle de colocar la URL entre comillas simples o dobles.

El comando jq

Aspectos técnicos ./jq

JSON es un formato de datos estructurados ampliamente utilizado que se utiliza normalmente en la mayoría de las API y servicios de datos modernos. Es particularmente popular en aplicaciones web debido a su naturaleza liviana y compatibilidad con JavaScript.

Desafortunadamente, shells como Bash no pueden interpretar y trabajar con JSON directamente. Esto significa que trabajar con JSON a través de la línea de comando puede ser engorroso e implica la manipulación de texto utilizando una combinación de herramientas como sed y grep

Allí es donde aparece jq, un potente procesador JSON para la consola.

Sintaxis básica

jq se basa en el concepto de filtros que funcionan sobre un flujo de JSON. Cada filtro toma una entrada y emite JSON a la salida estándar. Tomando el archivo JSON obtenido con cURL, una ejecución sencilla de jq nos devuelve todo el contenido del JSON.

```
edorio@DESKTOP-W10:~$ jq '.' resultado.json
```

Como vemos, no vamos a acceder a ningún atributo en especial, ya que con el modificador '.' no se lo indicamos

Accediendo a propiedades

Para poder acceder a una propiedad específica es necesario indicarla luego del punto, con el nombre de la misma

```
edorio@DESKTOP-W10:~$ jq '.display_name' resultado.json
```

En este caso, accederemos a la propiedad display_name del JSON. Si queremos acceder a varias propiedades, las separamos por coma

```
edorio@DESKTOP-W10:~$ jq '.display_name,.type' resultado.json
```

De esta manera, accedemos a display_name y type

TIP: Si alguna propiedad tuviese un espacio en su nombre, debemos envolverla con comillas dobles

Combinación de uso de ambos comandos

Combinar comandos con pipelines

Para poder combinar el poder de cURL con el recurso y la capacidad de proceso de jq debemos combinarlo usando pipelines. Para ello vamos a realizar una introducción a una de las características más interesantes que tiene la terminal. El pipeline o tubería es una función que permite utilizar la salida de un programa como entrada en otro.

El pipeline en Linux se representa con la barra vertical (|), la cual dividirá los comandos. Por ejemplo, si nosotros queremos saber la IP de nuestro equipo, lo hacemos con la instrucción:

```
edorio@DESKTOP-W10:~$ ip address
```

Esta nos devolverá muchísimos datos (MAC, protocolos, direcciones IPv4 e IPv6, entre otros). Si quisieramos filtrar dentro de ese texto por la cadena "192.168", lo deberíamos llevar a un archivo y allí buscar con grep.

```
edorio@DESKTOP-W10:~$ grep "192.168" miarchivo.txt
```

Pero es aquí en donde aparece la magia del pipeline, ya que podemos combinar ambas sentencias en una.

Para ello, primero colocamos nuestra sentencia inicial, sabiendo que tipo de salida puede tener, separamos con el pipeline y colocamos la segunda sentencia.

```
edorio@DESKTOP-W10:~$ ip address | grep "192.168"
```

Allí el grep nos indicará la línea coincidente con "192.168". Nuestro pipeline podría seguir aplicándose sin límites más allá de aquellos que imponga el sistema operativo, por ejemplo cantidad de procesos en ejecución

Aplicar el pipeline con cURL y jq

Conociendo el uso básico del pipeline, vamos a aplicarlo a la obtención de datos externos y el parseo de una propiedad específica, la cual la guardaremos en un archivo. Nuestro comando tendrá tres partes



Aplicar el pipeline con cURL y jq

En este caso, vemos cómo la sentencia obtiene el JSON con cURL, lo procesa con jq para obtener el display_name y el type, y finalmente lo guarda en un archivo llamado consultapipe.txt

```
edorio@DESKTOP-W10:~$ curl "https://nominatim.openstreetmap.org/reverse.php?lat=-34.60378&lon=-58.38161&zoom=18&format=jsonv2" | jq ".display_name,.type" | tee consultapipe.txt
```

[Ver PDF: comandos](#)

[Ver PDF: Ejercitación - Comandos basicos.pdf](#)

[Ver PDF: Ejercitacion](#)

Notas: Clase 4

Run time levels, capas de linux al arrancar Son estadios del kernel de Linux

Link: Run level

En el run level 5 repasaremos comandos de Linux

Para apagar la maquina desde la consola, hay que pasar por esos niveles. Primero hay que hacerlo desde el usuario **root**

Apaga el sistema

```
/sbin/init 0
```

Reinicia el sistema

```
/sbin/init 6
```

Para saber en que level estamos

```
who -r
```

Donde estamos ubicados

```
pwd
```

cat

concatena informacion de archivos y crea uno nuevo

cat nombrearchivo1 nombrearchivo2 > nombrearchivo3

tambien crea archivos

```
cat > lista_nombres
```

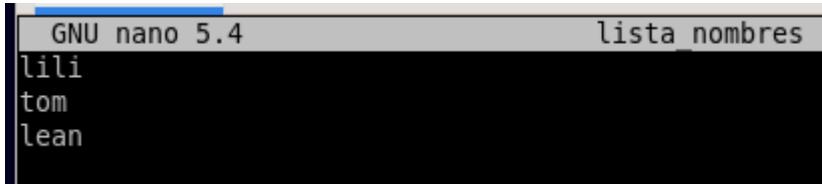
Se queda titilando esperando que lo guardo con **ctrl + D** y con **ctrl + x** salimos del archivo.

```
root@debiancliente:/home/liliana/infraPruebas# cat>lista_nombres
lili
tom
lean
root@debiancliente:/home/liliana/infraPruebas#
```

```
vagrant@server: $ cat>lista_nombres
dada
dada
Juan
Maria
vagrant@server: $ ls
lista_nombres
```

Lo consultamos con

```
nano lista_nombres
```



```
GNU nano 5.4                      lista_nombres
lili
tom
lean
```

con **control + x** salimos del archivo

con cat tambien podemos convertir un archivo a mayusculas o minusculas.

;

Ejecuta las cosas secuencialmente, primero uno, luego otro...

&&

Ejecuta si el segundo comando si el anterior fue exitoso, no usa la salida del comando anterior.

pipeline

conecta un comando con otro, devuelve un resultado/salida del comando que esta anteriormente usa la salida de un comando para otro comando. El pipeline, agarra el resultado del primer comando lo guarda en memoria y lo usa para el proximo comando

grep

Hace una busqueda y nos va a devolver lo que le pedimos que buscara, en este caso que traiga todas las ip que comiencen con: **192.168**

```
ip address | grep "192.168"
```

```
root@debiancliente:/home/liliana/infraPruebas# ip address | grep "192.168"
    inet 192.168.1.22/24 brd 192.168.1.255 scope global dynamic noprefixroute en
p0s3
root@debiancliente:/home/liliana/infraPruebas# █
```

Busca dentro de un archivo, o dentro de un resultado y devuelve el resultado pedido

```
root@debiancliente:/home/liliana/infraPruebas# grep lili lista_nombres
lili
root@debiancliente:/home/liliana/infraPruebas# grep l lista_nombres
lili
lean
root@debiancliente:/home/liliana/infraPruebas# █
```

sudo eleva permisos del superusuario

En debian no esta instalado por defecto, en ubuntu si con sudo en debian, no es necesario estar como root cuando tienes el sudo, ya que sudo eleva los permisos como superusuario; directamente se puede instalar:

eleva permisos de superusuario

```
sudo apt-get install curl
```

curl

Es una aplicacion consultar paginas web desde la terminal Consultar json, bajar un archivo.

Pagina que sirve para consultar comandos [cheat.sh](#)

Terminal 1

```
lili
lean
root@debiancliente:/home/liliana/infraPruebas# curl cheat.sh

The only cheat sheet you need
Unified access to the best
community driven documentation
repositories of the world

+-----+ +-----+ +-----+
| $ curl cheat.sh/ls      | | $ cht.sh btrfs      | | $ cht.sh lua/:learn |
| $ curl cht.sh/btrfs    | | $ cht.sh tar~list   | | Learn any* programming |
| $ curl cht.sh/tar~list  | |                         | | language not leaving |
| $ curl https://cht.sh   | |                         | | your shell
                           | | *) any of 60
+-----+ +-----+ +-----+
--- queries with curl ---+ + own optional client ---+ + learn, learn, learn! +-
+-----+ +-----+ +-----+
| $ cht.sh go/f<tab><tab> | | $ cht.sh --shell     | | $ cht.sh go zip lists |
| go/for      go/func      | | cht.sh> help       | | Ask any question using |
| $ cht.sh go/for          | | ...                 | | cht.sh or curl cht.sh: |
| ...           | |                         | | /go/zip+lists
|                         | |                         | | (use /,+ when curling)
+-----+ +-----+ +-----+
```

En esta pagina [cheat.sh](#) podemos consultar para que sirven comandos

```
root@debiancliente:/home/liliana/infraPruebas# curl cheat.sh/cd
cheat:cd
#Go to the given directory
cd path/to/directory

#Go to home directory of current user
cd

#Go up to the parent of the current directory
cd ..

#Go to the previously chosen directory
cd -

tldr:cd
# cd
# Change the current working directory.

# Go to the given directory:
cd path/to/directory

# Go to home directory of current user:
cd
```

chmod

Cambia los permisos de lectura, escritura y ejecucion Una forma facil de usar chmod es con sus valores numeros

usando la numeric representation: **rwx Read Write Execute** El primer digito habla sobre el dueño/usuario, el segundo es el grupo de usuarios(user group en linux: son los usuarios que estan dentro de un grupo de usuarios; los usuarios activos que estan dentro del servidor), y el tercero es el mundo; otros que no estan dentro del grupo.

```
# Numeric representations
7 - full (rwx)
6 - read and write (rw-)
5 - read and execute (r-x)
4 - read only (r--)
3 - write and execute (-wx)
2 - write only (-w-)
1 - execute only (--x)
0 - none (---)
```

Ej: Si le quiero dar permiso para todo:

```
chmod 777
```

Ej: Solo permiso de lectura

```
chmod 400
```

Ej: Archivo de lectura y ejecucion

```
chmod 505
```

ping

mide el tiempo de respuesta, se puede hacer con el tiempo de respuesta local o de otros servidores. Se usa para medir servidores de una aplicacion y queremos medir el tiempo de respuesta.

```
#Dice que mande solo 5 paquetes, que mida 5 veces el tiempo de respuesta
ping -c 5 "192.168..."
```

Abrir varias consolas a la misma vez

control alt f2 abre otra instancia **control alt f1** vuelve a la anterior

No paran, siguen funcionando en segundo plano

Actividad en clase

Listar servicios del sistema

Listar todos los servicios que se estan ejecutando en el sistema

```
systemctl list-unit-files --type service --all
```

UNIT FILE	STATE	VENDOR PRESET
accounts-daemon.service	enabled	enabled
alsa-restore.service	static	-
alsa-state.service	static	-
alsa-utils.service	masked	enabled
anacron.service	enabled	enabled
apache-htcacheclean.service	disabled	enabled
apache-htcacheclean@.service	disabled	enabled
apache2.service	enabled	enabled
apache2@.service	disabled	enabled
apparmor.service	enabled	enabled
apt-daily-upgrade.service	static	-
apt-daily.service	static	-
autovt@.service	alias	-
avahi-daemon.service	enabled	enabled
bluetooth.service	enabled	enabled
bolt.service	static	-
colord.service	static	-
configure-printer@.service	static	-
console-getty.service	disabled	disabled
console-setup.service	enabled	enabled

Pararemos el apache2 que habiamos instalado

```
sudo systemctl stop apache2
```

accounts-daemon.service	disabled
apache-htcacheclean.service	disabled
apache-htcacheclean@.service	disabled
apache2.service	enabled
apache2@.service	disabled
apparmor.service	enabled
apt-daily-upgrade.service	static

Prendiendo de nuevo apache

```
sudo systemctl start apache2
```

Para sacar a apache del inicio, cuando se activa la vm

```
sudo systemctl disable apache2
```

Para activar el servicio de apache desde el inicio

```
sudo systemctl enable apache2
```

Para ver el status, si esta corriendo o no

```
sudo systemctl status apache2
```

Analizando los datos que van y vienen

Analizar los datos que van y vienen de nuestros servidores

netstat esta deprecado

ss -plnt

ss -plnt muestra los puertos que estan abierto escuchando la maquina

```
root@debiancliente:/home/liliana# ss -plnt
State      Recv-Q      Send-Q      Local Address:Port      Peer Address:Port
Process
LISTEN      0          128          0.0.0.0:22          0.0.0.0:*
  users:(("sshd",pid=467,fd=3))
LISTEN      0          128          [::]:22            [::]:*
  users:(("sshd",pid=467,fd=4))
root@debiancliente:/home/liliana#
```

Para borrar paquetes: **apt-get remove nombredeloaquelquieroborrar**

tcpdump

Sirve para monitorear un puerto que este siendo utilizado por un servicio Se usa para analizar y monitorear paquetes de red Monitorea el trafico de red de un determinado puerto

```
tcpdump -i enp0s3 port 80
```

puede cambiar el nombre, lo verificamos con `ip address`

```
root@debiancliente:/home/liliana# ip address
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 08:00:27:f6:7a:d0 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.22/24 brd 192.168.1.255 scope global dynamic noprefixroute enp0s3
        valid_lft 84698sec preferred_lft 84698sec
    inet6 fe80::a00:27ff:fed0:64 scope link noprefixroute
        valid_lft forever preferred_lft forever
root@debiancliente:/home/liliana#
```

C5 - Shell Scripting - Parte II

Scripting

Automatizar procesos de tareas al sistema

En esta clase vamos a presentar la interfaz de Bash como un shell de Unix, donde el usuario puede programar que se ejecuten órdenes al sistema operativo. Bash nos sirve para comunicarnos con nuestro sistema. Un ejemplo claro es la forma de automatizar la orden de realizar un backup cada determinado tiempo de la base de datos.

Ver video: Introducción a bash

Interacción con sistema operativo de una forma automatizable

`bash` es una interfaz que interpreta las órdenes que el usuario le hace al sistema ejecutado en una consola de UNIX. También se pueden leer y ejecutar órdenes desde un archivo llamado script que nos permite hacer persistente una lista de tareas a realizar por el sistema operativo. También es un lenguaje de scripting, lo que lo hace una herramienta muy potente para la administración de sistemas y automatización de tareas.

Para ejecutar múltiples comandos en un solo paso desde el shell, podemos escribirlos en una sola línea y separarlos en ;

Creamos un nuevo archivo

```
touch myscript
nano myscript
```

Shebang: #!

Define qué shell usaremos, `bash` es el shell en nuestro caso

```
#!/bin/bash
```

Los comandos de shell se escriben 1 por linea

Tipos de variables

Globales o de entorno

El sistema Linux establece algunas variables de entorno globales cuando iniciamos sesión en nuestro sistema y siempre son en LETRAS MAYÚSCULAS. Si queremos ver las variables de entorno que estamos usando y que están cargadas en nuestra sesión, escribimos el comando **printenv** o **env** en nuestro shell. Para declarar una variable global se debe realizar de la siguiente manera:

```
{} export NOMBREVARIABLE=valor
```

Para acceder a la misma utilizamos la sentencia

```
{} $NOMBREVARIABLE
```

De usuario o locales

Las variables del tipo usuario o local tienen la particularidad que pueden ser accedidas solo por el usuario y la sesión en la que fueron creadas. Una variable local se declara de la forma sencilla:

```
{} nombrevariable=valor
```

Para acceder a la misma utilizamos la sentencia

```
{} $nombrevariable
```

Bash scripting

¡Ahora sí! Manos a la obra. En este apartado vamos a tener nuestros primeros scripting realizados por nosotros. De a poco nos introduciremos en las estructuras de control de los algoritmos de programación.

[Ver PDF: Estructuras de control](#)

Estructuras de control

Sentencia if-then

Los scripts de Bash necesitarán condicionales. Normalmente nos imaginamos un escenario tal como "Si el valor es menor que 10, hacé esto, si no hacé aquello". Para ello utilizamos if-then. La estructura más básica de la sentencia **if-then** es así:

if command; then hacer algo fi

```
#!/bin/bash
if whoami; then
```

```
echo "It works"  
fi
```

Sentencia if-else

La sentencia **if-then-else** toma la siguiente estructura:

if comand; then hacer algo else hacer otra cosa fi

Si el comando se ejecuta y retorna cero (lo cual significa éxito), no ejecuta los comandos después de la sentencia else. Por otro lado, si la sentencia if retorna un número distinto de cero (lo cual significa que la condición no se cumple), el shell ejecuta los comandos después de la sentencia else. En el ejemplo que sigue vemos cómo nos devuelve un mensaje de acuerdo si el comando ping fue satisfactorio.

```
#!/bin/bash  
ping -c 1 8.8.8.8  
if [ $? -ne 0 ]; then  
echo "No está en red"  
else  
echo "Sí está en red"  
fi
```

Comparaciones numéricas

Podemos realizar una comparación numérica entre dos valores numéricos utilizando las sentencias de esta tabla

number1 -eq number2	Comprueba si number1 es igual a number2.
number1 -ge number2	Comprueba si number1 es más grande o igual number2.
number1 -gt number2	Comprueba si number1 es más grande que number2.
number1 -le number2	Comprueba si number1 es más pequeño o igual number2.
number1 -lt number2	Comprueba si number1 es más pequeño que number2.
number1 -ne number2	Comprueba si number1 no es igual a number2.

Teniendo en cuenta que la sentencia de comparación se encuentra entre corchetes, exemplificamos:

```
#!/bin/bash  
num = 11  
if [ $num -gt 10 ]; then  
echo "$num is bigger than 10"  
else  
echo "$num is less than 10"  
fi
```

Comparaciones de cadenas

Podemos realizar una comparación de cadena entre dos valores alfanuméricos utilizando las sentencias de esta tabla

`string1 = string2` Comprueba si string1 es idéntico a string2. `string1 != string2` Comprueba si string1 no es idéntico a string2. `string1 < string2` Comprueba si string1 es menor que string2. `string1 > string2` Comprueba si string1 es mayor que string2. `-n string1` Comprueba si string1 es mayor que cero. `-z string1` Comprueba si string1 tiene una longitud de cero.

Podemos aplicar la comparación de cadenas en nuestro ejemplo. Si estamos logueados como root, va por el **if**, si no por el **then**

```
#!/bin/bash
user="root"
if [ $user = $USER ]; then
echo "The user $user is the current logged in user"
fi
```

Cálculos matemáticos

Podemos realizar cálculos matemáticos básicos utilizando la sintaxis \$ ((2 + 2)):

```
#!/bin/bash
var1=$(( 5 + 5 ))
echo $var1
var2=$(( $var1 * 2 ))
echo $var2
```

Ver PDF: Ejemplo en Bash.pdf

ver PDF: Realizamos nuestro primer script.pdf

Notas: Clase 5

Entro desde powershell a mi maquina virtual de debian

```
PS C:\Users\Clickon> ssh liliana@192.168.1.22
liliana@192.168.1.22's password:
Linux debiancliente 5.10.0-8-amd64 #1 SMP Debian 5.10.46-4 (2021-08-03) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Wed Sep 15 22:32:09 2021 from 192.168.1.10
liliana@debiancliente:~$ su root
Contraseña:
root@debiancliente:/home/liliana#
```

Abrimos

```
nano script.sh
```

```
liliana@debiancliente: ~/Documentos/liliana
GNU nano 5.4          script.sh *
#!/bin/bash
for word in $(cat lista_nombres)
do
  echo "El nombre es $word"
done
```

Pegamos el siguiente código

```
#!/bin/bash
for word in $(cat lista_nombres)
do
  echo "El nombre es $word"
done
```

Lo guardamos.

Modificamos los permisos de ejecución del archivo desde el usuario **root** con el código

```
chmod +x script.sh
```

Para ejecutar un escript en bash

```
./script.sh
```

```
liliana@debiancliente:~/Documentos/liliana$ su root
Contraseña:
root@debiancliente:/home/liliana/Documentos/liliana# chmod +x script.sh
root@debiancliente:/home/liliana/Documentos/liliana# ./script.sh
El nombre es Lean
El nombre es Lili
El nombre es Tom
El nombre es Silvia
El nombre es Margarita
El nombre es Cristina
El nombre es Erik
El nombre es Julieta
El nombre es Mario
El nombre es Raynar
El nombre es Juan
El nombre es Rocco
El nombre es Emilio
El nombre es Maria
root@debiancliente:/home/liliana/Documentos/liliana#
```

El siguiente script agregandole la sentencia if recorrera todo el listado escribiendo El nombre es Lean y con la sentancia if se ejecutara el comando Encontre a juan cuando el ciclo llegue a Juan

```
#!/bin/bash
for word in $(cat lista_nombres)
do
    echo "El nombre es $word"
    if [ $word = "Juan" ]; then
        echo "Encontre a $word"
    fi
done
```

control o guardo, enter, control x salgo.

```
root@debiancliente:/home/liliana/Documentos/liliana# ./script.sh
El nombre es Lean
El nombre es Lili
El nombre es Tom
El nombre es Silvia
El nombre es Margarita
El nombre es Cristina
El nombre es Erik
El nombre es Julieta
El nombre es Mario
El nombre es Raynar
El nombre es Juan
Encontre a Juan
El nombre es Rocco
El nombre es Emilio
El nombre es Maria
root@debiancliente:/home/liliana/Documentos/liliana#
```

Ejercicio 2

Instalamos jq en git bash y la vm

jq sirve para consultar datos Json

modificamos de nuevo el script, con este script **jq '.gender'** va a leer, **read -r gen** va a leer y **echo** va a devolver/mostrar el genero.

```
#!/bin/bash
for word in $(cat lista_nombres)
do
    echo "El nombre es $word"
    curl -s https://api.genderize.io/?name=$word | jq '.gender' |{ read -r gen;
    echo "Gender of $word is: $gen"; }
    curl -s https://api.nationalize.io/?name=$word | jq '.country[0].country_id' |
{ read -r cn; echo "Country of $word is: $cn"; }
done
```

Resultado:

```
root@debiancliente:/home/liliana/Documentos/liliana# ./script.sh
El nombre es Lean
Gender of Lean is: "male"
Country of Lean is: "AR"
El nombre es Lili
Gender of Lili is: "female"
Country of Lili is: "GP"
El nombre es Tom
Gender of Tom is: "male"
Country of Tom is: "CZ"
El nombre es Silvia
Gender of Silvia is: "female"
Country of Silvia is: "SK"
```

C6 - Repaso

Repasso acceso a VM desde powershell

```
PS C:\Users\david> ssh davidpigna@192.168.1.80
davidpigna@192.168.1.80's password:
Linux debiancliente 5.10.0-8-amd64 #1 SMP Debian 5.10.46-4 (2021-08-03) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Sep 23 13:59:39 2021
davidpigna@debiancliente:~$ su root
Contraseña:
root@debiancliente:/home/davidpigna# ■
```

Notas clase 6

Powershell funciona tambien como lenguaje de programacion

[Ver PDF: Armemos un script un poco mas complejo.pdf](#)

Desarrollo actividad con git bash

usamos una lista de nombres 456 nombres de github

[Link github](#)

creamos una carpeta

```
david@DESKTOP-KKF3N7P MINGW64 ~/Desktop
$ cd ..

david@DESKTOP-KKF3N7P MINGW64 ~
$ mkdir ejerjq

david@DESKTOP-KKF3N7P MINGW64 ~
$ cd ejerjq

david@DESKTOP-KKF3N7P MINGW64 ~/ejerjq
$ nano script.sh
```

entramos con nano a editar el nuevo script `script.sh` copiamos de la actividad el codigo al nano y tabulamos

Código actividad a reemplazar

Incluirías y/o adaptarías a tu script. Reemplazá el texto en rojo con la dirección web del archivo.

```
req=`curl <urlAllListado> | shuf`  
  
countA=0  
  
for nombre in $req  
do  
    if [[ $nombre == A* ]] && [ $countA -le 4 ] ;  
    then  
        echo $nombre  
        let "countA++"  
    fi  
done
```

shuf es un comando que hace referencia a shuffle, elige un solo nombre aleatoriamente

```
david@DESKTOP-KKF3N7P MINGW64 ~/ejerjq  
$ curl cheat.sh/shuf  
% Total    % Received % Xferd  Average Speed   Time     Time     Time  Current  
          Dload  Upload Total   Spent   Left  Speed  
100  805  100  805    0     0  1468      0 --:--:-- --:--:-- --:--:-- 1468# shuf  
# Generate random permutations.  
  
# Randomize the order of lines in a file and output the result:  
shuf filename  
  
# Only output the first 5 entries of the result:  
shuf -n 5 filename  
  
# Write the output to another file:  
shuf filename -o output_filename  
  
# Generate random numbers in range:  
shuf -i 1-10
```

```
GNU nano 5.8                                     script.sh  
req= curl https://raw.githubusercontent.com/olea/lemarios/master/nombres-propios-es.txt | shuf  
countA=0  
  
for nombre in $req  
do  
    if [[ $nombre == A* ]] && [ $countA -le 4 ] ;  
    then  
        echo $nombre  
        let "countA++"  
    fi  
done
```

reemplazamos:

si el comando anterior fue exitoso ejecuta el segundo, **a*** indica que empieza con a y sigue con cualquier cosa, dice 4 porque empieza en 0

```
[[ $nombre == A* ]] && [ $countA -le 4 ]
```

Ejecutamos el script y selecciono 5 nombres con A

```
david@DESKTOP-KKF3N7P MINGW64 ~/ejerjq
$ ./script.sh
% Total    % Received % Xferd  Average Speed   Time   Time   Time  current
          Dload  Upload Total Spent   Left Speed
100  3555  100  3555    0     0 12141      0 --::-- --::-- --::-- 12216
Aarón
Alicia
Adón
Alipio
Ambrosio
```

Los espacios entre los corchetes son necesarios, **le** les equal

El ejercicio pide consultar las apis, hay que acomodar el curl

El curl va despues del let

```
GNU nano 5.8                                     script.sh
req= curl https://raw.githubusercontent.com/olea/lemarios/master/nombres-propios-es.txt | shuf
countA=0

for nombre in $req
do
    if [[ $nombre == A* ]] && [ $countA -le 4 ] ;
    then
        echo $nombre
        let "countA++"
        curl -s https://api.genderize.io/?name=$nombre | jq '.gender' | { read -r gen; echo "Gender of $nombre is: $gen" }
        curl -s https://api.nationalize.io/?name=$nombre | jq '.country[0].country_id' | { read -r cn; echo "Country of $nombre is: $cn" }
    fi
done
```

con este script consultara las apis

```
david@DESKTOP-KKF3N7P MINGW64 ~/ejerjq
$ ./script.sh
% Total    % Received % Xferd  Average Speed   Time   Time   Time  Current
          Dload  Upload Total Spent   Left Speed
100  3555  100  3555    0     0 51226      0 --::-- --::-- --::-- 51521
Andrea
Gender of Andrea is: "male"
Country of Andrea is: "SM"
Alejandro
Gender of Alejandro is: "male"
Country of Alejandro is: "AR"
Ananias
Gender of Ananias is: null
Country of Ananias is: null
Ascención
Gender of Ascención is: null
Country of Ascención is: null
Abrahán
Gender of Abrahán is: null
Country of Abrahán is: null
```

Ejercicio pide: seleccionar aleatoriamente 5 nombres que no empiecen ni con a ni con l

ponemos un **elif**; otra condicion del **if** y una opcion es poner que sea distinto de a

```
GNU nano 5.8                                     script.sh
req= curl https://raw.githubusercontent.com/olea/lemarios/master/nombres-propios-es.txt | shuf
countA=0
countotro=0
for nombre in $req
do
    if [[ $nombre == A* ]] && [ $countA -le 4 ] ;
    then
        echo $nombre
        let "countA++"
        curl -s https://api.genderize.io/?name=$nombre | jq '.gender' | { read -r gen; echo "Gender of $nombre is: $gen" }
        curl -s https://api.nationalize.io/?name=$nombre | jq '.country[0].country_id' | { read -r cn; echo "Country of $nombre is: $cn" }
    elif [[ $nombre != A* ]] && [[ $nombre != L* ]] && [ $countotro -le 4 ] ;
    then
        echo $nombre
        let "countotro++"
        curl -s https://api.genderize.io/?name=$nombre | jq '.gender' | { read -r gen; echo "Gender of $nombre is: $gen" }
        curl -s https://api.nationalize.io/?name=$nombre | jq '.country[0].country_id' | { read -r cn; echo "Country of $nombre is: $cn" }
```

Lo testeamos...

```
david@DESKTOP-KKF3N7P MINGW64 ~/ejerjq
$ ./script.sh
% Total    % Received % Xferd  Average Speed   Time   Time     Time  Current
          Dload  Upload Total   Spent   Left  Speed
100  3555  100  3555    0      0 15484      0 ---:--- ---:--- ---:--- 15524
olga
Gender of olga is: "female"
Country of olga is: "MD"
Albina
Gender of Albina is: "female"
Country of Albina is: "UZ"
Rosa
Gender of Rosa is: "female"
Country of Rosa is: "AD"
Mar
Gender of Mar is: "female"
Country of Mar is: "ES"
Noelia
Gender of Noelia is: "female"
Country of Noelia is: "ES"
David
Gender of David is: "male"
Country of David is: "US"
Adela
Gender of Adela is: "female"
Country of Adela is: "BA"
Aniano
```

[Ver script.sh](#)

el codigo es:

```
req=`curl https://raw.githubusercontent.com/olea/lemarios/master/nombres-propios-es.txt | shuf` 
countA=0
countotro=0
for nombre in $req
do
  if [[ $nombre == A* ]] && [ $countA -le 4 ] ;
  then
    echo $nombre
    let "countA++"
    curl -s https://api.genderize.io/?name=$nombre | jq '.gender' | { read -r gen; echo "Gender of $nombre is: $gen"; }
    curl -s https://api.nationalize.io/?name=$nombre | jq '.country[0].country_id' | { read -r cn; echo "Country of $nombre is: $cn"; }
  elif [[ $nombre != A* ]] && [[ $nombre != L* ]] && [ $countotro -le 4 ] ;
  then
    echo $nombre
    let "countotro++"
    curl -s https://api.genderize.io/?name=$nombre | jq '.gender' | { read -r gen; echo "Gender of $nombre is: $gen"; }
    curl -s https://api.nationalize.io/?name=$nombre | jq '.country[0].country_id' | { read -r cn; echo "Country of $nombre is: $cn"; }
  fi
done
```

Este mismo ejercicio lo podemos hacer con Powershell pero lleva otra sintaxis

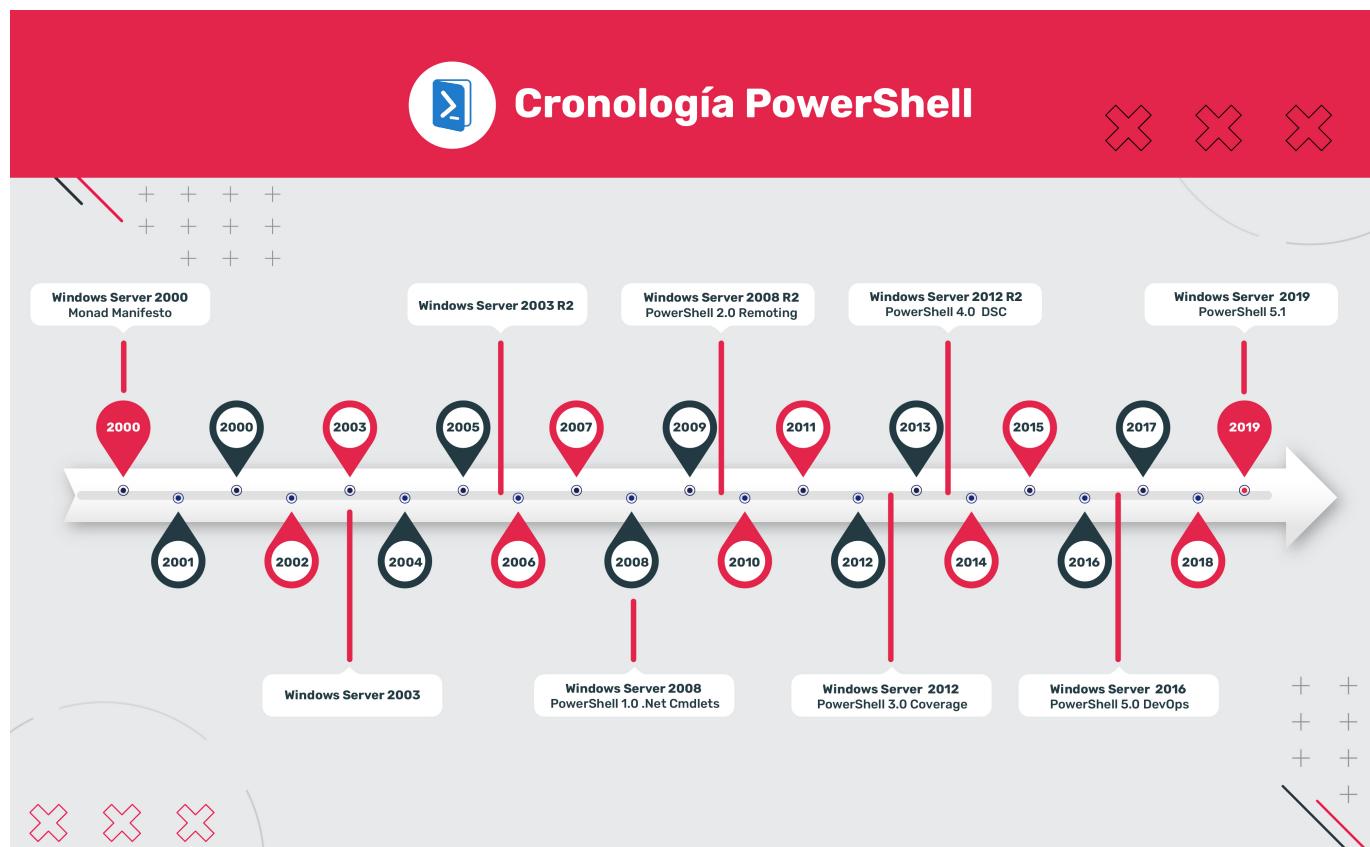
```
$req = Invoke-WebRequest -Method Get -Uri<urlAllListado>$nombresConA =
$req.Content.split("`n") | Where-Object { $_ -like"A*" }
1..5 | ForEach-Object {
    $random = Get-Random -Minimum 0 -Maximum $($nombresConA.count-
1)$nombresConA[$random]
}
```

C7 - PowerShell

Introducción a PowerShell

[Ver video: PowerShell](#)

Te introducimos al lenguaje PowerShell. Sigamos el recorrido con un poco de historia.



Seguimos recorriendo los conceptos de PowerShell. A continuación veremos la diferencia entre PowerShell y PowerShell Core.

Diferencias entre PowerShell y PowerShell Core

¿Qué es PowerShell Core? ¿Y cuál es la diferencia con Windows PowerShell?

- PowerShell Core es una nueva versión de PowerShell,
- Un shell de línea de comandos y lenguaje de scripts que se incluye con Microsoft Windows.

Es decir que contamos con dos ediciones de PowerShell. Existe el PowerShell de hace una década que está integrado en todas las versiones recientes del sistema operativo Windows de Microsoft y el nuevo PowerShell

Core.

Microsoft ve a PowerShell Core como una evolución de PowerShell. El primero está disponible como una aplicación multiplataforma, el último solo para Windows.

La naturaleza multiplataforma de PowerShell Core significa que las secuencias de comandos que se escriban se ejecutarán en cualquier sistema operativo compatible. Se pueden escribir scripts de PowerShell Core en Windows y usarlos en dispositivos macOS X o Linux compatibles. Incluso hay versiones experimentales (no compatibles) para dispositivos ARM.

Microsoft trabaja activamente en PowerShell Core. PowerShell, por otro lado, se encuentra en un estado que se puede comparar mejor con el soporte extendido para las versiones de Windows. Microsoft no tiene planes de agregar funciones a PowerShell, pero lanzará correcciones de errores críticos y actualizaciones de seguridad.

Esto significa que el riesgo de regresión será muy bajo para Windows PowerShell, por lo que se puede contar con él como una plataforma estable para cargas de trabajo existentes.

Por último, vale la pena aclarar que PowerShell Core no afecta a Windows PowerShell de ninguna manera en dispositivos con Windows. Por lo que ambos pueden convivir sin problemas.

Guías de instalación de PowerShell

A continuación encontrarás la guía de instalación de PowerShell para cada sistema operativo.

Windows

Linux

macOS

Usos comunes de PowerShell

 Link: Usos comunes de PowerShell

Monitoreo

Para escribir scripts que pueden ser utilizados por un software de monitoreo.

Testear infraestructura

Las pruebas de infraestructura en Pester son código de PowerShell que ejecuta el módulo Pester PowerShell y se crea de una manera específica, conocida como lenguaje específico de dominio (DSL). Este DSL describe el estado deseado y tiene el código necesario para verificar ese estado y comparar el resultado.

Automatizacion de procesos

Automatizar tareas en un proceso de liberación de software (CI/CD).

Automatizacion de tareas

PowerShell sirve para facilitar a los administradores de sistemas las tareas de automatización, administración y configuración de sistemas.

Configuracion management

Al utilizar PowerShell DSC.

Comandos cmdlet

[Ver PDF: comandos cmdlet.pdf](#)

Cmdlet	Descripción.
Set-Location	Establece la ubicación de trabajo actual en una ubicación específica.
Get-Content	Obtiene el contenido del elemento en la ubicación especificada.
Add-Content	Agrega contenido a los elementos especificados, por ejemplo, agregar palabras a un archivo.
Set-Content	Escribe o reemplaza el contenido de un elemento con contenido nuevo.
Copy-Item	Copia un elemento de una ubicación a otra.
Remove-Item	Elimina los elementos especificados.
Move-Item	Mueve un elemento de una ubicación a otra.
Set-Item	Cambia el valor de un elemento al valor especificado en el comando.
New-Item	Crea un nuevo elemento.
Start-Job	Inicia un trabajo en segundo plano de Windows PowerShell.
Compare-Object	Compara dos conjuntos de objetos.
Group-Object	Agrupa objetos que contengan el mismo valor en propiedades especificadas.
Invoke-WebRequest	Obtiene contenido de una página web en Internet.
Measure-Object	Calcula los valores de propiedad de determinados tipos de objetos. Puede realizar tres tipos de mediciones según los parámetros del comando.
Resolve-Path	Resuelve los caracteres comodín en una ruta y muestra el contenido.
Resume-Job	Reinicia un trabajo suspendido.
Set-Variable	Establece el valor de una variable. Crea la variable si no existe una con el nombre solicitado.

Cmdlet	Descripción.
Show-Command	Muestra comandos de Windows PowerShell en una ventana de comandos gráfica.
Sort-Object	Ordena los objetos por valores de propiedad.
Start-Service	Inicia uno o más servicios detenidos.
Start-Process	Inicia uno o más procesos en la computadora local.
Suspend-Job	Detiene temporalmente un trabajo.
Wait-Job	Espera a que los trabajos en segundo plano de Windows PowerShell se completen antes de mostrar el símbolo de sistema.
Where-Object	Selecciona los objetos del conjunto de objetos que se le pasan.
Write-Output	Envía el objeto especificado al siguiente comando en la canalización. Si es el último comando en la canalización, el objeto se mostrará en consola.

Ejemplo uso comandos cmdlet

- Set-Location: Establece la ubicación actual. Entra a los registros de la maquina

```
PS C:\> Set-Location -Path "HKLM:\"
PS HKLM:\>
```

- Get-Content: Obtiene el contenido de un archivo de texto.

paralelismo con bash es `cat`

```
Get-Content -Path .\LineNumbers.txt

Esta es la línea 1
Esta es la línea 2
```

- Add-Content: Añade contenido a un archivo de texto.

```
Add-Content -Path .\LineNumbers.txt -Value 'Fin de archivo'

Esta es la línea 1
Esta es la línea 2
Fin de archivo
```

- Set-Content: Modifica una línea de un archivo de texto.

```
(Get-Content -Path .\LineNumbers.txt) |  
ForEach-Object {$_.Replace('Fin de archivo', 'Esta es la línea 3')}  
  
Set-Content -Path .\LineNumbers.txt  
  
Get-Content -Path .\LineNumbers.txt  
Esta es la línea 1  
Esta es la línea 2  
Esta es la línea 3
```

- Copy-Item: Copia un archivo a un directorio específico.

```
Copy-Item "C:\Wabash\Logfiles\mar1604.log.txt" -Destination "C:\Presentation"
```

- Remove-Item: Este ejemplo elimina de la carpeta actual todos los archivos que tienen una extensión de nombre de archivo .doc con la excepción de los que tengan un 1 en su nombre.

```
Remove-Item * -Include *.doc -Exclude *1*
```

- Move-Item: Mueve un archivo a otro directorio y lo renombra.

```
Move-Item -Path C:\test.txt -Destination E:\Temp\tst.txt
```

- Set-Item: Este ejemplo crea un alias llamado np para el Notepad de Windows.

```
Set-Item -Path alias:np -Value "c:\windows\notepad.exe"
```

- New-Item: Crea un archivo en el directorio actual.

```
New-Item -Path . -Name "archivo.txt" -ItemType "file" -Value "Texto de prueba"
```

- Start-Job: Ejecuta un script como un proceso en segundo plano.

```
Start-Job -FilePath C:\Scripts\Sample.ps1
```

- Compare-Object: Compara el contenido de dos archivos:

Este ejemplo compara el contenido de dos archivos, el ejemplo usa los siguientes dos archivos de texto, cada uno con un valor en una línea diferente.

- verduras.txt contiene los valores: papa, zapallo, tomate.
- frutas.txt contiene los valores: banana, tomate, naranja. El resultado muestra solo las líneas que son diferentes entre los archivos. En verduras.txt

Usa el objeto de referencia (<=) y en frutas.txt usa el objeto de diferencia (=>). Las líneas que se encuentran en los dos archivos no son mostradas.

```
Compare-Object -ReferenceObject (Get-Content -Path C:\Test\verduras.txt)
-DifferenceObject (Get-Content -Path C:\Test\frutas.txt)

InputObject SideIndicator
-----
banana =>
naranja =>
papa <=
zapallo <=
```

- Group-Object: Agrupa archivos por su extensión.

```
$files = Get-ChildItem -Path $PSHOME -Recurse
$files | Group-Object -Property extension -NoElement | Sort-Object -
Property Count -Descending

Count Name
-----
365 .xml
231 .cdxml
197
169 .ps1xml
142 .txt
114 .psd1
63 .psm1
49 .xsd
36 .dll
15 .mfl
15 .mof
```

- Invoke-WebRequest: Muestra una página web.

```
Invoke-WebRequest -Uri "http://www.google.com"
```

- Measure-Object: Este comando cuenta los archivos y carpetas del directorio actual.

```
Get-ChildItem | Measure-Object
```

- Resolve-Path: Muestra la ruta actual.

```
PS C:\> Resolve-Path ~
```

```
Path
```

```
----
```

```
C:\Users\User
```

- Resume-Job: Reanuda un trabajo por nombre.

```
PS C:\> Resume-Job -Name WorkflowJob, InventoryWorkflow, WFTest*
```

- Set-Variable: Establece una variable y obtiene su valor.

```
Set-Variable -Name "desc" -Value "Una descripción"
Get-Variable -Name "desc"
```

```
Name Value
```

```
---- ----
```

```
desc Una descripción
```

- Show-Command: Abre un cmdlet en una ventana de comandos.

```
Show-Command -Name "Invoke-Command"
```



- Sort-Object: Ordena el directorio actual por nombre.

```
PS> Get-ChildItem -Path . | Sort-Object
```

```
Directory: C:\Users\User
```

```
Mode LastWriteTime Length Name
```

```
---- ----- ----- -----
```

```
-a---- 2/13/2019 08:55 26 anotherfile.txt
-a---- 2/13/2019 13:26 20 Bfile.txt
-a---- 2/12/2019 15:40 118014 Command.txt
-a---- 2/1/2019 08:43 183 CreateTestFile.ps1
d---- 2/25/2019 18:25 Files
```

```
d---- 2/25/2019 18:24 Logs
-ar--- 2/12/2019 14:31 27 ReadOnlyFile.txt
-a---- 2/12/2019 16:24 23 Zsystemlog.log
```

- Start-Service: Inicia un servicio detenido.

```
Start-Service -Name "eventlog"
```

- Start-Process: Inicia un proceso que use valores por defecto.

```
Start-Process -FilePath "sort.exe"
```

Kit de supervivencia de PowerShell

Kit de supervivencia de PowerShell

Lo utilizamos para obtener ayuda e información sobre cmdlets, módulos y otros.

Get-Help

1

Lo utilizamos para buscar un cmdlet.

Get-Command

3

Lo utilizamos para obtener información sobre los módulos disponibles en el sistema.

Get-Command

5

Update-Help



Lo utilizamos para actualizar el repositorio de ayuda.



Get-Member



Lo utilizamos para mostrar las propiedades y los métodos de un objeto devuelto.



\$PSVersionTable



Lo utilizamos para ver la variable de entorno que contiene información sobre PowerShell.

[Ver PDF: Kit de supervivencia de PowerShell](#)

¿Qué es un pipeline?

Es una serie de comandos conectados mediante operadores de canalización ("|", ASCII 124). Cada operador del pipeline envía los resultados del comando anterior al siguiente comando. La salida del primer comando se puede enviar para procesar como entrada para el segundo comando y el resultado se puede enviar a otro

comando. Lo anterior es una cadena de comandos o pipeline complejo que se compone de una serie de comandos simples.

```
Command-1 | Command-2 | Command-3
```

En este ejemplo, los objetos que Command-1 emite se envían a Command-2 . Command-2 procesa los objetos y los envía a Command-3. Este último procesa los objetos y los envía a través del pipeline. Dado que no hay más comandos en la canalización, los resultados se muestran en la consola. En un pipeline, los comandos se procesan en orden de izquierda a derecha. El procesamiento se controla como una operación única y el resultado se muestra a medida que se genera.

¿Qué es una variable?

Una variable es una unidad de memoria en la que se almacenan los valores. En PowerShell, las variables se representan mediante cadenas de texto que comienzan por un signo de dólar (\$), como \$a, \$process o \$my_var.

Los nombres de variable no distinguen mayúsculas de minúsculas y pueden incluir espacios y caracteres especiales. Sin embargo, los nombres de variable que incluyen caracteres especiales y espacios son difíciles de usar y deben evitarse. Para obtener más información, vea nombres de variable que incluyen caracteres especiales.

Tipos de variables en PowerShell

- Variables creadas por el usuario: el usuario crea y mantiene las variables. De forma predeterminada, las variables que se crean en la línea de comandos de PowerShell solo existen mientras la ventana de PowerShell está abierta. Cuando se cierra la ventana de PowerShell, se eliminan las variables. Para guardar una variable, deberán agregarla a su perfil de PowerShell.
- Variables automáticas: estas almacenan el estado de PowerShell. Estas variables las crea PowerShell y cambia sus valores según sea necesario para mantener su precisión. Los usuarios no pueden cambiar el valor de estas variables.
- Variables de preferencia: estas almacenan las preferencias de usuario para PowerShell. Estas variables las crea PowerShell y se rellenan con valores predeterminados. Los usuarios pueden cambiar sus valores.

¿Qué son las estructuras de control?

Las estructuras de control permiten modificar el flujo de ejecución de las instrucciones de un programa. Con ellas se puede:

- De acuerdo con una condición, ejecutar un grupo u otro de sentencias (If-Then-Else).
- De acuerdo con el valor de una variable, ejecutar un grupo u otro de sentencias (Switch-Case).
- Ejecutar un grupo de sentencias solo cuando se cumpla una condición (Do-While).
- Ejecutar un grupo de sentencias hasta que se cumpla una condición (Do-Until).

- Ejecutar un grupo de sentencias un número determinado de veces (For-Next).

Tipos de estructuras de control

Todas las estructuras de control tienen un único punto de entrada. Las estructuras de control se pueden clasificar en: secuenciales, iterativas y de control avanzadas. Esta es una de las cosas que permiten que la programación se rija por los principios de la programación estructurada.

Los lenguajes de programación modernos tienen estructuras de control similares. Básicamente lo que varía entre las estructuras de control de los diferentes lenguajes es su sintaxis. Cada lenguaje tiene una sintaxis propia para expresar la estructura

¿Qué es un script de PowerShell?

Un script es un archivo de texto sin formato que contiene uno o más comandos de PowerShell. Los scripts de PowerShell tienen una extensión de archivo .ps1.

Ejecutar un script es muy parecido a ejecutar un cmdlet. Pueden ejecutar scripts en el equipo o en una sesión remota en otro equipo.

Al escribir un script se guarda un comando para su uso posterior y facilita el uso compartido con otros usuarios. Lo más importante es que permite ejecutar los comandos simplemente escribiendo la ruta de acceso del script y el nombre de archivo. Los scripts pueden ser tan simples como un solo comando en un archivo o tan complejos como un programa completo.

¿Qué es una función en PowerShell?

En PowerShell, como en muchos lenguajes, una función es un conjunto de instrucciones a las que les damos un nombre. El principal interés de las funciones es que podemos llamarlas varias veces, sin tener que volver a escribir las instrucciones en cada llamada.

Si se escriben scripts o comandos únicos de una línea de PowerShell y encuentran que a menudo tienen que modificarlos para distintos escenarios, es bastante probable que estos sean buenos candidatos para convertirlos en una función que se pueda volver a usar.

[Ver PDF: Definición Módulo de Powershell V2.pdf](#)

módulos de PowerShell

¿Qué es un módulo?

Como mencionamos antes, un módulo es un paquete que contiene objetos de PowerShell, como cmdlets, proveedores, funciones, flujos de trabajo, variables y alias. Los objetos u acciones de este paquete se pueden implementar en un script de PowerShell, una DLL compilada o una combinación de ambos. Por lo general, estos archivos se agrupan en un solo directorio. Las personas que escriben comandos pueden usar módulos para organizar sus comandos y compartirlos con otros. Las personas que escriben módulos pueden agregar los comandos en los módulos a sus sesiones de PowerShell y usarlos como comandos integrados.

Carga automática de los módulos

A partir de PowerShell 3.0, este importa módulos automáticamente la primera vez que ejecuta cualquier comando en un módulo instalado. Ahora podés usar los comandos en un módulo sin ninguna configuración o

configuración de perfil, por lo que no es necesario administrar los módulos después de instalarlos en tu computadora. Los comandos de un módulo también son más fáciles de encontrar. El cmdlet Get-Command obtiene todos los comandos en todos los módulos instalados, incluso si aún no están en la sesión. Podés encontrar un comando y usarlo sin tener la necesidad de importar el módulo primero. Solo los módulos que se almacenan en la ubicación especificada por la variable de entorno PSModulePath se importan automáticamente. Los comandos que incluyen un carácter comodín (*) se consideran para descubrimiento, no para uso y no importan ningún módulo. Los módulos en otras ubicaciones deben importarse ejecutando el Import-Module cmdlet.

[Ver PDF: Trabajando con módulos de PowerShell.pdf](#)

¿Cómo utilizar e instalar un módulo de PowerShell?

- Instalar el módulo.
- Buscar los comandos que agregó el módulo.
- Utilizar los comandos que agregó el módulo

Si recibís un módulo como una carpeta con archivos, debés instalarlo en tu computadora antes de poder usarlo en PowerShell. La mayoría de los módulos se instalan automáticamente. PowerShell viene con varios módulos preinstalados, a veces llamados módulos centrales. En equipos con Windows, si las funciones que se incluyen con el sistema operativo tienen cmdlets para administrarlas, esos módulos están preinstalados.

Módulos de PowerShell El cmdlet Install-Module obtiene uno o más módulos que cumplen los criterios especificados de un repositorio en línea. Este verifica que los resultados de la búsqueda sean módulos válidos y copia las carpetas del módulo en la ubicación de instalación. Los módulos instalados no se importan automáticamente después de la instalación. Podés filtrar qué módulo está instalado en función de las versiones mínima, máxima y exacta de los módulos especificados.

Si el módulo que se está instalando tiene el mismo nombre o versión —o contiene comandos en un módulo existente—, se muestran mensajes de advertencia.

¿Cómo encontrar módulos instalados?

Para encontrar módulos que están instalados en una ubicación de módulo predeterminada, pero que aún no se han importado a tu sesión, debés escribir:

```
Get-Module -ListAvailable
```

Para encontrar los módulos que ya se han importado a tu sesión, escribí en tu sesión de PowerShell:

```
Get-Module
```

El cmdlet **Find-Module** busca módulos en un repositorio que coinciden con los criterios especificados. Find-Module devuelve un objeto **PSRepositoryItemInfo** para cada módulo que encuentra. Los objetos se pueden enviar por el pipeline a cmdlets como **Install-Module**.

La primera vez que Find-Module intente utilizar un repositorio, es posible que te solicite que instales actualizaciones. Si la fuente del repositorio no está registrada con el cmdlet Register-PSRepository, se devuelve un error. Find-Module devuelve la versión más reciente de un módulo si no se utilizan parámetros que limiten la versión.

```
Find-Module Name PowerShellGet
Version      Name          Repository           Description
-----      ----          -----           -----
2.1.0       PowerShellGet   PSGallery          PowerShell module
with commands for discovering...
```

¿Cómo buscar los comandos en un módulo?

Usamos el comando Get-Command cmdlet para buscar todos los comandos disponibles. Podés usar los parámetros del Get-Command cmdlet para filtrar comandos por módulo, nombre y sustantivo

```
Get-Command -Module <module-name>
```

¿Cómo obtener ayuda para los comandos de un módulo?

Si el módulo contiene archivos de ayuda para los comandos que exporta, el Get-Help cmdlet mostrará los temas de ayuda. Usá el mismo Get-Help formato de comando que usariás para obtener ayuda para cualquier comando de PowerShell. Para obtener ayuda para un comando de un módulo, escribí:

```
GetHelp <command-name>
```

Para obtener ayuda en línea para el comando en un módulo, escribí:

```
GetHelp <command-name>
```

Para descargar e instalar los archivos de ayuda de los comandos de un módulo, escribí:

```
Update-Help -Module <module-name>
```

¿Cómo importar un módulo o un archivo del mismo?

La importación es necesaria cuando un módulo no está instalado en las ubicaciones especificadas por la variable de entorno PSModulePath, \$env:PSModulePath; o el módulo consta de un archivo, como un archivo .dll o .psm1, en lugar de un módulo típico que se entrega como una carpeta. Para importar módulos, podés usar el Import-Module cmdlet. Para importar módulos en una ubicación PSModulePath en la sesión actual, utilizá el siguiente formato de comando:

```
Import-Module <module-name>
```

¿Cómo quitar un módulo?

Al quitar un módulo, se eliminan de la sesión los comandos que el módulo agregó. Para hacerlo tendrás que utilizar el siguiente formato de comando:

```
Remove-Module <module-name>
```

Notas clase 7

Repaso:

Guardar variables en powershell y mostrar con `echo`

```
PS C:\Users\david> $nombre = "david"
PS C:\Users\david> print $nombre
No se puede encontrar el archivo david
PS C:\Users\david> echo $nombre
david
```

Powershell es una herramienta para automatizar SO, servicios, guardar consultas de sql, es un lenguaje de programacion

Paralelismo con bash La diferencia entre powershell y bash, se usan soluciones propias de powershell, usa comandos internos. En vez de usar apps exteriores, se usan comandos propios.

[Ver PDF: Realizamos nuestros primeros scripts en PowerShell.pdf](#)

Powershell es un lenguaje de programacion

- ver setup de mysql con powershell

[Ver PDF: Trabajamos_con_modulos_de_PowerShell.pdf](#)

Solucion de actividad

Creamos la base de datos desde el CLI de MySQL UNICODE pide contrasenia, ponemos la que es de mysql

```

mysql> CREATE DATABASE test;
Query OK, 1 row affected (0.02 sec)

mysql> Use test;
Database changed
mysql> CREATE TABLE test (field1 INT)
-> ;
Query OK, 0 rows affected (0.09 sec)

mysql> ALTER TABLE
-> test
-> CONVERT TO CHARACTER SET utf8mb4
-> COLLATE utf8mb4_unicode_ci;
Query OK, 0 rows affected (0.02 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> INSERT INTO test (field1) VALUES (1);
Query OK, 1 row affected (0.01 sec)

mysql> EXIT_

```

Como administrador en PowerShell:

Buscamos modulo:

```
find-module -name invokequery
```

```

PS C:\WINDOWS\system32> find-module -name invokequery■

Version          Name           Repository      Description
-----          ----           -----
1.0.1           InvokeQuery    PSGallery     Query any database!

```

Instalo el modulo

```
install-module -name invokequery
```

Invoke-MySQLQuery

Nos va a permitir hacer consultas sql desde powershell, es un programa preprogramado que lo bajamos como un modulo.

una **Ventaja** puede ser que se puedan automatizar consultas. Disponible para usar como en un **script.sh**

<https://powerwamp.readthedocs.io/en/latest/Invoke-MySQLQuery/>

En PowerShell, dentro de una variable que se va a llamar **\$creds**

```
$creds = Get-Credential
```

Esto nos va a preguntar usuario y contraseña de nuestra base de datos y la va a guardar en esa variable los datos de acceso.

Guarda dentro de la variable `$query` la búsqueda de la base de datos.

```
$query = New-SqlQuery -sql "select * from test"
```

consulta base de datos por un módulo que fue instalado

```
Invoke-MySqlQuery -SqlQuery $query -Credential $creds -Server localhost -Database test
```

Resultado:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Clickon> $creds = Get-Credential

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
PS C:\Users\Clickon> $query = New-SqlQuery -sql "select * from test"
PS C:\Users\Clickon> Invoke-MySqlQuery -SqlQuery $query -Credential $creds -Server localhost -Database test

field1
-----
 1

PS C:\Users\Clickon>
```

Otros comandos:

Para saber si un comando está instalado:

```
Get-command New-SqlQuery
```

Para saber dónde están instalados los módulos, puede mostrar varias rutas.

```
$env:PSModulePath
```

ver pdf paso Ana: Powershell-cheatsheet.pdf

C8 - Python

Introducción a Python

Python es un lenguaje de escritura rápida, escalable, robusto y de código abierto. Esto hace que Python sea un aliado perfecto para la inteligencia artificial, ya que permite plasmar ideas complejas con unas pocas líneas de código, lo que no es posible con otros lenguajes.

Te invitamos a conocer este lenguaje del que todos hablan.

[Ver video: Introducción a Python](#)

1. gratuito
2. open source
3. Nuevas librerías constantemente
4. Lenguaje multiparadigma
5. Apto para todas las plataformas

Historia de Python

Su nombre se debe a la afición de Van Rossum al grupo Monty Python. Su concepción se enfocaba en que fuera fácil de usar y aprender, sin que esto penalizara sus capacidades. En su momento, no llegó a adquirir la suficiente importancia debido a la falta de recursos en el hardware de la época.



¿Qué es y para qué sirve Python?

Python se define como un "lenguaje de programación versátil, multiplataforma y multiparadigma que se destaca por su código legible y limpio". A su vez, se emplea en plataformas de alto tráfico como Google, YouTube o Facebook. Su principal objetivo es la automatización de procesos, lo que hará de las tareas algo mucho más simple. Uno de sus puntos fuertes es que "comprueba los errores sobre la marcha". En este sentido, Python crea un código con gran legibilidad, que ahorra tiempo y recursos.

Usos de Python

Python se convirtió en uno de los lenguajes de programación más utilizados en la actualidad, aunque tiene una presencia en el mercado de más de 30 años. Uno de sus principales aliados es la inteligencia artificial, pero también se destaca en aplicaciones web y big data.

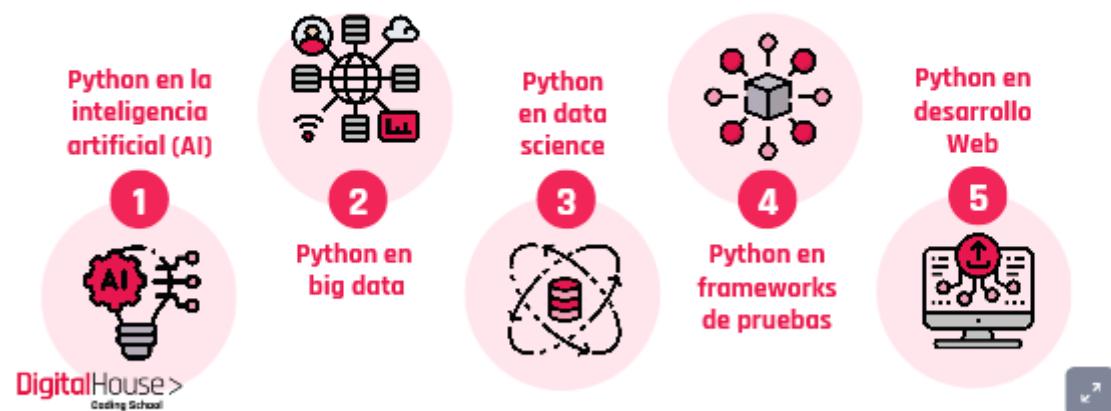
Python también se utiliza para obtener información de otros sitios web, lo que comúnmente se denomina "scraping". Incluso tiene aplicaciones prometedoras en el campo médico que mejoran la capacidad de brindar diagnósticos y tratamientos precisos a los pacientes. Otras áreas como astronomía, robótica, vehículos autónomos, negocios, meteorología y desarrollo de interfaces gráficas de usuario también se benefician con el uso de Python.

Avancemos para conocer por qué es importante Python en el contexto de infraestructura.

Importancia de python en infraestructura

¿Por qué es importante saber Python en el contexto de infraestructura?

Principales industrias que usan este lenguaje de programación:



Importancia de python en infraestructura

Python en la inteligencia artificial (AI)

Python es un lenguaje de escritura rápido, escalable, robusto y de código abierto, ventajas que hacen de Python un aliado perfecto para la inteligencia artificial.

Permite plasmar ideas complejas con unas pocas líneas de código, lo que no es posible con otros lenguajes.

Python en big data

El uso de Python está muy extendido en el análisis de datos y la extracción de información útil para empresas

Python en data science

Python se ocupa de los datos tabulares, matriciales y estadísticos, e incluso los visualiza con bibliotecas populares.

Python en frameworks de pruebas

Python es ideal para validar ideas o productos, ya que tiene muchos frameworks integrados que ayudan a depurar el código y ofrecen flujos de trabajo y ejecución rápidos.

Python en Desarrollo Web

Python permite construir mucho más con menos líneas de código, por lo que se crean prototipos de forma más eficiente.

Pros y Contras

Para qué se utiliza Python

- Para el desarrollo de aplicaciones:
- Desarrollo web
- Videojuegos
- Análisis de datos
- Internet de las cosas (IoT)

Fue usado para desarrollar:

- Instagram
- YouTube
- Google

Pros

- Aprendizaje:** Es un lenguaje relativamente fácil de aprender.
- Académico:** Lenguaje preferido por las principales universidades en el mundo para comenzar a programar.
- Librerías:** Cuenta con una extensa cantidad de módulos (escritos en C) como apoyo.

Contras

- Rendimiento:** Su rendimiento aún no es comparable con el de C/C++.
- Desarrollo móvil:** Muy pocas aplicaciones móviles son desarrolladas con Python. Esta no es un área fuerte del lenguaje.
- Errores:** Algunos errores se pueden detectar solamente en tiempo de ejecución.

Python es uno de los lenguajes de programación oficiales en Google.

Python es un lenguaje interpretado de propósito general. Su diseño acentúa la legibilidad del código y su sintaxis permite expresar conceptos en pocas líneas de código, a diferencia de lenguajes como C++ o Java. Python soporta múltiples paradigmas de programación.

Continuemos este recorrido. Ya vimos para qué se utiliza el lenguaje, dónde es mayormente aplicado y los beneficios que tenemos al utilizarlo. Seguiremos ahora con la parte práctica, metiéndonos en profundidad en los términos y conceptos de este lenguaje. ¿Nos acompañás?

Nos ponemos a practicar

Ver pdf: Variables en Python.pdf

Variables

¿Qué es una variable en Python?

En algunos lenguajes de programación, las variables se pueden entender como "cajas" en las que se guardan los datos, pero en Python las variables son "etiquetas" que permiten hacer referencia a los datos (que se guardan en unas "cajas" llamadas objetos).

Python es un lenguaje de programación orientado a objetos y su modelo de datos también está basado en objetos.

Para cada dato que aparece en un programa, Python crea un objeto que lo contiene.

Cada objeto tiene:

- **Un identificador único** (un número entero, distinto para cada objeto). El identificador permite a Python referirse al objeto sin ambigüedades.
- **Un tipo de datos** (entero, decimal, cadena de caracteres, etc.). El tipo de datos permite saber a Python qué operaciones pueden hacerse con el dato.
- **Un valor** (el propio dato)

¿Cómo definir una variable?

Las variables en Python se crean cuando se definen por primera vez, es decir, cuando se les asigna un valor por primera vez. Para asignar un valor a una variable se utiliza el operador de igualdad (`=`). A la izquierda de la igualdad se escribe el nombre de la variable y a la derecha el valor que se quiere dar a la variable.

Una variable puede almacenar números, texto o estructuras más complicadas (que se verán más adelante).

Si se va a almacenar texto, el texto debe escribirse entre comillas simples (`'`) o dobles (`"`), que son equivalentes.

A las variables que almacenan texto se les suele llamar cadenas (de texto). Si no se escriben comillas, Python supone que estamos haciendo referencia a otra variable.

```
numeros = 000000
numeros1 = 02473
numeros2 = 12653
text = 'variabletexto'
text2 = 'variabletexto'
```

Nombres de variables

Aunque no es obligatorio, se recomienda que el nombre de la variable esté relacionado con la información que se almacena en ella, para que sea más fácil entender el programa. Si el programa es trivial o mientras se está escribiendo un programa, esto no parece muy importante, pero si se consulta un programa escrito por otra persona o escrito por uno mismo, pero hace tiempo, resultará mucho más fácil entender el programa si los nombres están bien elegidos. También se acostumbra a utilizar determinados nombres de variables en algunas ocasiones, como se verá más adelante, pero esto tampoco es obligatorio.

Tipos de variables

En el apartado anterior hay definiciones de algunos de los tipos de variables que hay en Python:

números decimales, números enteros y cadenas (una o más letras).

Aunque se definan de forma similar, para Python no es lo mismo un número entero, un número decimal o una cadena ya que, por ejemplo, dos números se pueden multiplicar, pero dos cadenas no (**curiosamente, una cadena sí que se puede multiplicar por un número**).

Una vez se ha definido una variable, se puede utilizar para hacer cálculos o para definir nuevas variables, como muestran los siguientes ejemplos:

Utilizar o modificar variables ya definidas

```
>>>a = 2
>>>a + 35
>>>horas = 5
>>>minutos = 60 * horas
>>>segundos = 60 * minutos
>>>segundos
1800
>>>horas = 1
>>>minutos = 2
>>>segundos = 3
>>>segundos + 60 * minutos + 3600 * horas
3723
```

En caso de utilizar una variable no definida anteriormente, Python genera un mensaje de error.

```
>>>dias = 7 * semanas
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    dias = 7 * semanas
NameError: name 'semanas' is not defined
```

[Ver pdf: Estructuras de control en Python.pdf](#)

Indentación (sangría)

En Python, las líneas de código que están dentro de un mismo deben estar agrupadas, teniendo el mismo número de espacios a la izquierda de cada línea, al igual que sucedería en la vida real. A modo de ejemplo:

- Carrefour
 - Carnicería
 - Cerdo
 - Pollo
 - Pescadería

Este siguiente caso no sería correcto, y en Python generaría un error (o el funcionamiento no sería el esperado):

- Frutería
 - Peras
 - Manzanas

Lógicamente, **Manzanas** no puede estar al mismo nivel que **Frutería**. En Python, se recomienda usar siempre bloques de cuatro espacios, aunque si se usan otro número de espacios, también funcionará. También se pueden usar tabuladores, aunque se recomienda no usarlos.

if en Python

En todo programa llega el momento, que en función de una determinada condición, hay que realizar una serie de cosas u otra.

Esto se hace con el comando **if** (condición principal), con los opcionales **elif** (condiciones adicionales, se pueden poner tantas como se quiera) y **else** (si no se ha cumplido ninguna de las anteriores, solo se puede poner una vez y al final).

A modo de ejemplo:

```
>>> Alonso_Position=1>
>> if (Alonso_Position==1):
>>>     print("Espectacular Alonso, se ha hecho justicia a pesar del coche")
>>>     print("Ya queda menos para ganar el mundial")
>>> elif (Alonso_Position>1):
>>>     print("Gran carrera de Alonso, lástima que el coche no esté a la altura")
>>> else:
>>>     print("No ha podido terminar la carrera por una avería mecánica")
```

Condiciones en Python

Las condiciones que se suelen usar con más frecuencia son:

Condicion	Descripción
------------------	--------------------

a == b	Indica si a es igual a b
--------	--------------------------

a < b	a > b		not		NO: Niega la condición que le sigue.	and		Y: Junta dos condiciones que tienen que cumplirse las dos		or		O: Junta dos condiciones y tienen que cumplirse alguna de las dos.
-------	-------	--	-----	--	--------------------------------------	-----	--	---	--	----	--	--

Bucles While y for

while

En ocasiones, tenemos que repetir varias veces una determinada tarea hasta conseguir nuestro objetivo. En Python esto se realiza con el comando **while**. Con los while, hay que tener la precaución de no realizar un

«bucle infinito», que consiste en un bucle que nunca termina por un error en la programación. En el caso siguiente, esto ocurriría si no hubiéramos puesto la línea `vuelta=vuelta+1`.

A modo de ejemplo while en Python se usa así:

```
>>> while vuelta<10:  
>>>     print("Vuelta "+str(vuelta))  
>>>     vuelta=vuelta+1  
Vuelta 1  
Vuelta 2  
Vuelta 3  
Vuelta 4  
Vuelta 5  
Vuelta 6  
Vuelta 7  
Vuelta 8  
Vuelta 9
```

for

En ocasiones, tenemos que repetir varias veces una determinada tarea hasta conseguir nuestro objetivo. En Python esto se realiza con el comando `for`. En el caso del for, no es posible realizar un bucle infinito. Como se puede ver en el siguiente ejemplo, range genera una secuencia de números desde 1 hasta 10. `for` se puede utilizar con cualquier objeto con el que se pueda iterar (ir saltando de elemento en elemento).

A modo de ejemplo for en Python se usa así:

```
>>> for vuelta in range(1,10):  
>>>     print("Vuelta "+str(vuelta))  
Vuelta 1  
Vuelta 2  
Vuelta 3  
Vuelta 4  
Vuelta 5  
Vuelta 6  
Vuelta 7  
Vuelta 8  
Vuelta 9
```

`for` se puede utilizar con cualquier objeto con el que se pueda iterar (ir saltando de elemento en elemento), como vemos en este ejemplo con una lista:

```
>>> coches = ('Ferrari', 'Tesla', 'BMW', 'Audi')  
>>> for coche in coches:  
>>>     print(coche)  
Ferrari  
Tesla
```

```
BMW  
Audi
```

Si lo combinamos con la función enumerate, además irá dándole un número a cada elemento:

```
>>> coches = ('Ferrari', 'Tesla', 'BMW', 'Audi')  
>>> for i, coche in enumerate(coches):  
>>>     print(str(i) + " - " + coche)  
0 - Ferrari  
1 - Tesla  
2 - BMW  
3 - Audi
```

[Ver pdf: Trabajamos con Estructuras de Control II](#)

Como hemos observado anteriormente, los algoritmos requieren dos estructuras de control importantes: iteración y selección. Ambas están disponibles en Python en varias formas. El programador puede elegir la instrucción que sea más útil para la circunstancia dada.

Para la iteración, Python proporciona una instrucción `while` estándar y una instrucción `for` muy potente.

La instrucción `while` repite un cuerpo de código mientras una condición sea verdadera.

Por ejemplo:

Ejemplos estructuras de control

Ejemplo while

Este código imprime la frase “Hola, mundo” cinco veces. **La condición en la instrucción while se evalúa al inicio de cada repetición.** Si la condición es `True`, el cuerpo de la instrucción se ejecutará.

Es fácil ver la estructura de una instrucción `while` de Python debido al patrón de sangrado obligatorio que el lenguaje impone.

```
>>> contador = 1  
>>> while contador <= 5:  
...     print("Hola, mundo")  
...     contador = contador + 1  
  
Hola, mundo  
Hola, mundo  
Hola, mundo  
Hola, mundo  
Hola, mundo
```

La instrucción `while` es una estructura iterativa de propósito general que usaremos en una serie de algoritmos diferentes. En muchos casos, una condición compuesta controlará la iteración. Un fragmento como el

siguiente:

```
while contador <= 10 and not hecho:
```

Haría que el cuerpo de la instrucción fuera ejecutado solo en el caso en que se cumplan ambas partes de la condición.

El valor de la variable `contador` tendría que ser **menor o igual a 10** y el valor de la variable `hecho` tendría que ser **False (not False es True)** de modo que True and True da como resultado True.

Aunque este tipo de estructura es muy útil en una amplia variedad de situaciones, otra estructura iterativa, la instrucción `for`, se puede usar junto con muchas de las colecciones de Python.

Ejemplo for

La instrucción `for` puede usarse para iterar sobre los miembros de una colección, siempre y cuando la colección sea una secuencia.

Por ejemplo:

```
>>> for item in [1,3,6,2,5]:  
    print(item)  
  
1  
3  
6  
2  
5
```

Asigna a la variable `item` cada valor sucesivo de la lista `[1,3,6,2,5]`. Entonces se ejecuta el cuerpo de la iteración. Esto funciona para cualquier colección que sea una secuencia (listas, tuplas y cadenas).

Un uso común de la instrucción `for` es implementar una iteración definida sobre un rango de valores.

Ejecutará la función `print` cinco veces. La función `range` devolverá un objeto de rango que representa la secuencia "0,1,2,3,4" y cada valor se asignará a la variable `item`. Este valor es entonces elevado al cuadrado y se imprime en pantalla.

La siguiente instrucción:

```
>>> for item in range(5):  
...     print(item**2)  
...  
0  
1  
4  
9
```

```
16
```

```
>>>
```

instrucciones de selección

Las instrucciones de selección les permiten a los programadores hacer preguntas y luego, con base en el resultado, realizar diferentes acciones.

La mayoría de los lenguajes de programación proporcionan dos versiones de esta útil estructura: ifelse e if.

Un ejemplo sencillo de una selección binaria que utiliza la instrucción ifelse es el siguiente:

En este ejemplo, el objeto referenciado por n se **comprueba para ver si es menor que cero**. Si es así, se imprime un mensaje indicando que es negativo. Si no es así, la instrucción ejecuta la cláusula else y calcula la raíz cuadrada.

```
if n<0:  
    print("Lo siento, el valor es negativo")  
else:  
    print(math.sqrt(n))
```

estructuras de selección

Las estructuras de selección, como ocurre con cualquier otra estructura de control, pueden anidarse de modo que el resultado de una pregunta ayude a decidir si se pregunta la siguiente.

Por ejemplo, supongamos que puntaje es una variable que contiene una referencia a un puntaje de un examen de ciencias de la computación.

Este fragmento clasificará un valor llamado puntaje mediante la impresión de la calificación cualitativa obtenida. **Si el puntaje es mayor o igual a 90**, la instrucción imprimirá "A". **Si no lo es (else)**, se hace la pregunta siguiente. **Si el puntaje es mayor o igual a 80**, entonces debe estar entre 80 y 89, ya que la respuesta a la primera pregunta era falsa. En este caso se imprimirá la letra "B".

Puede verse que el patrón de sangrado de Python ayuda a dar sentido a la asociación entre if y else sin necesidad de elementos sintácticos adicionales.

```
if puntaje >= 90:  
    print('A')  
else:  
    if puntaje >= 80:  
        print('B')  
    else:  
        if puntaje >= 70:  
            print('C')  
        else:
```

```
if puntaje >= 60:  
    print('D')  
else:  
    print('F')
```

Una sintaxis alternativa para este tipo de selección anidada utiliza la palabra clave `elif`. El `else` y el `if` siguiente se combinan para eliminar la necesidad de niveles de anidamiento adicionales. Tené en cuenta que el último `else` sigue siendo necesario para proporcionar el caso por defecto, en caso de que todas las demás condiciones fallen.

```
if puntaje >= 90:  
    print('A')  
elif puntaje >= 80:  
    print('B')  
elif puntaje >= 70:  
    print('C')  
elif puntaje >= 60:  
    print('D')  
else:  
    print('F')
```

estructura de selección de una sola vía

Python también tiene una estructura de selección de una sola vía, la instrucción `if`. Con esta instrucción, si la condición es verdadera, se realiza una acción. En caso de que la condición sea falsa, el procesamiento simplemente continúa con la instrucción que siga después del `if`. Por ejemplo, el siguiente fragmento **comprobará primero si el valor de una variable n es negativo**. Si lo es, entonces es modificado mediante la función de valor absoluto. Sea cual sea el caso, la siguiente acción es calcular la raíz cuadrada.

```
if n<0:  
    n = abs(n)  
print(math.sqrt(n))
```

comprensión de listas

Regresando a las listas, existe un método alternativo para crear una lista que usa estructuras de iteración y de selección, conocido como comprensión de listas.

Una comprensión de lista permite crear fácilmente una lista basada en algunos criterios de procesamiento o de selección.

Por ejemplo, si quisieramos crear una lista de los primeros 10 cuadrados perfectos, podríamos usar una instrucción `for`:

La variable x toma los valores de 1 a 10 especificados por la estructura for. El valor de $x*x$ se calcula y se agrega a la lista que se está construyendo.

```
>>> listaCuadrados=[]
>>> for x in range(1,11):
    listaCuadrados.append(x*x)

>>> listaCuadrados
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
```

Usando comprensión de listas, podemos hacer lo mismo en un único paso como

```
>>> listaCuadrados=[x*x for x in range(1,11)]
>>> listaCuadrados
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
>>>
```

La sintaxis general para una comprensión de listas también permite agregar un criterio de selección para que únicamente se agreguen ciertos ítems.

Esta comprensión de listas construyó una lista que contenía solamente los cuadrados de los números impares en el rango de 1 a 10.

Por ejemplo:

```
>>> listaCuadrados=[x*x for x in range(1,11) if x%2 != 0]
>>> listaCuadrados
[1, 9, 25, 49, 81]
>>>
```

Cualquier secuencia que soporte la iteración se puede utilizar dentro de una comprensión de listas para construir una nueva lista.

```
>>>[letra.upper() for letra in 'estructuras' if letra not in 'aeiou']
['S', 'T', 'R', 'C', 'T', 'R', 'S']
>>>
```

Notas clase 8

Toma valores aleatorios entre 0 1 y 2, el ultimo numero no lo toma:

```
random.randrange(0,3)
```

Las variables dentro de python son un objeto

Concatena de un mismo tipo con `+`, con llaves tambien Concatena con distintos tipos de dato con `,`

C9 - Repaso

Notas clase 9

Repasso vagranFile

```
Vagrant.configure("2") do |config|
  # El nombre de la virtual machine, el nombre externo
  config.vm.define "server" do |server|
    config.vm.box = "debian/buster64"
    # El nombre del host; la maquina en si, el nombre interno
    server.vm.hostname = "server"
    # Define si es una red privada o publica, en este caso es publica, con conexion
    # a internet
    server.vm.network "public_network"
    # Hypervisor que vamos a usar para ejecutar la vm
    config.vm.provider "virtualbox" do |vb|
      vb.gui = true
      # Si piden poner 1g escribimos 1024, si son 2gb 2048; siempre va de 512 en 512
      vb.memory = "512"
      # Instala el apache
      server.vm.provision "shell", inline: <<-SHELL
        apt-get update
        apt-get install -y apache2
      SHELL
      # Mueve un file y lo reemplaza por otro html
      server.vm.provision "file", source: "index.html", destination: "index.html"
      server.vm.provision "shell", inline: "mv index.html /var/www/html/index.html"
    end
  end
```

Podemos automatizar consultas, inicios de servidores, consultas a bases de datos

C10 - Configuration Management

Ver pdf: ¿Qué es configuration management.pdf

Configuration management y change management son dos de los procesos fundamentales del conjunto de metodologías conocido como ITIL

Change management

ITIL describe la gestión de cambios como el proceso de controlar y gestionar un cambio a lo largo de todo su ciclo de vida con el objetivo de minimizar el riesgo.

¿Qué es un cambio?

Según ITIL, un cambio es la modificación o eliminación de cualquier cosa que pueda afectar directa o indirectamente los servicios. Básicamente, cualquier cambio en la infraestructura de IT de una organización puede afectar las operaciones de la organización.

Ejemplos de un cambio: reemplazo de hardware, instalación de software en un servidor o la modificación de una configuración de un sistema que alterará su comportamiento

Configuration management

Es el proceso que permite gestionar los cambios de configuración de nuestros activos informáticos —ya sean de software o hardware—, permitiendo a la organización mantener un registro histórico y a su vez aplicar controles —como un proceso de aprobación para cambios que cumplan con determinadas características—. Cada uno de los activos informáticos en el contexto de este proceso se los conocen como **configuration items (CI)** y se almacenan en lo que es llamado CMDB (configuration management database).

Change management + Configuration management

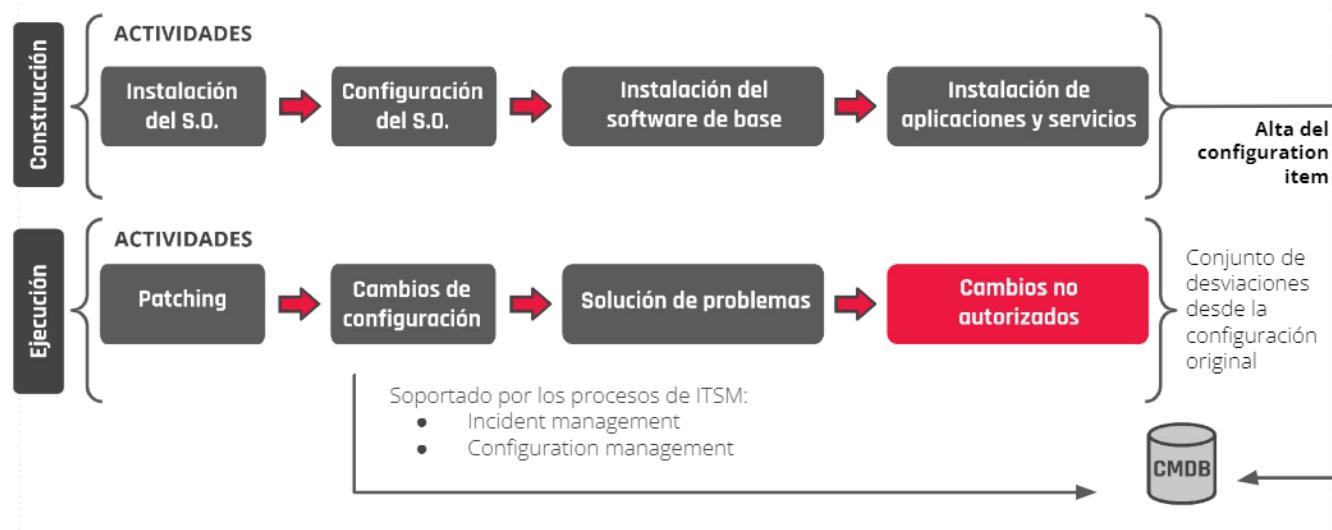
Ambos procesos se complementan, ya que **change management** aporta a la gobernabilidad de los cambios que se efectúan en el parque tecnológico de una organización, y configuration management nos permite gestionar una base de datos (CMDB) con la información de nuestros activos (CIs) y un historial de los cambios realizados para cada uno de ellos. De esta manera, de ocurrir la falla de uno de nuestros activos que nos fuercen a restaurarlo, por ejemplo, una falla crítica en un servidor, el proceso de configuration management nos aporta una vista de todos los cambios sucedidos para ese activo desde su creación, y así permitirnos reproducir cada uno de los cambios y devolverlo al estado anterior a la falla

Configuration management tradicional

¿Cómo se ve en la práctica?

Tomemos como ejemplo el ciclo de vida de un servidor. Este se puede dividir en 2 etapas:

- La etapa de **construcción** del servidor: un conjunto de pasos que pueden ser automatizados, documentados o sencillamente conocimiento común del área de IT, que permiten la puesta en marcha de un servidor y registrarlo en la CMDB como un nuevo CI.
- Su ciclo de vida, a lo que podemos llamar etapa de **ejecución**: todo lo que suceda en relación al activo y cambios que puedan suceder en él.



Etapa de construcción de un servidor

Instalación del S.O.

El proceso de instalación del sistema operativo en el hardware o máquina virtual.

Configuración del S.O.

El conjunto de configuraciones definido por la organización como requeridas para el que el servidor pueda formar parte del ecosistema

Instalación del software de base

Es probable que la organización de IT haya determinado que debe instalarse software adicional para que el servidor pueda operar correctamente dentro del ecosistema. Estos pueden incluir: antivirus, software de IDS / IPS, agentes de backup, etcétera

Instalación de aplicación y servicios

Finalmente, durante el proceso de construcción del servidor se debe instalar y configurar el software o servicio que dará sentido a su existencia. Por ejemplo, si el servidor está destinado a ser un servidor web, no solo se trata de instalar el servidor web (IIS, NGINX, Apache, etc.), sino que también se trata de desplegar el sitio web que lo alojará.

Etapa de ejecución de un servidor

Patching

El proceso de instalar actualización del sistema operativo, dependiendo del sistema operativo es la cadencia con la que este proceso se ejecutará. En términos generales las organizaciones eligen ejecutar instalaciones mensuales de actualizaciones. Este proceso debería verse reflejado mediante el proceso de change management.

Cambios de configuración

Cualquier cambio de configuración del sistema operativo o hardware que responda a una necesidad. Estos deberían verse reflejados como parte del proceso de change management.

Solución de problemas

Cualquier incidencia que haya producido una falla en el sistema reduciendo o anulando su funcionamiento. Requiriendo la intervención de un administrador para restaurar el nivel de servicio. Estas deberían verse reflejadas como parte del proceso de incident management.

Cambios no autorizados

¡Alerta! Cualquier cambio o actividad realizado en el servidor y que no queda documentado en un ticket, ya sea como un cambio o un incidente.

Problemáticas: Configuration management + Change management, ¿es practicable?

¿Es posible reconstruir un activo el mismo estado que se encontraba previo a la falla que nos forzó a restaurarlo? El poder realizar esto de manera exitosa depende de muchas variables:

- La disciplina que hayamos tenido para realmente reflejar cada uno de los cambios realizados al activo durante su ciclo de vida.
- Quedará a discreción del administrador a cargo de la restauración que cambios son necesarios reproducir, y cuales pueden ser obviados.
- Error humano, podemos equivocarnos al intentar reconstruir el activo. Toda tarea manual conlleva lugar a errores. Y cuanto más antiguo sea un activo, mayor será la cantidad de cambios que tengamos que aplicar, en consecuencia, mayor la posibilidad de cometer un error.
- La existencia de cambios no autorizados

Un cambio no autorizado que es implementado con éxito es más **peligroso** que uno que no lo es. Ya que produce una configuración efectiva sin dejar evidencia de la misma

¿Mascotas vs. Ganado?

Cuando se habla de configuration management se tiene a hacer una divergencia entre 'Mascotas' y 'Ganado'. ¿Por qué?

En términos generales, se dice que en los procesos tradicionales, aquellos en los que encontramos las problemáticas previamente mencionadas, cada servidor administrado de forma individual se convierte en una mascota, con un trato personalizado.

Mientras que en los procesos modernos de administración de servidores, los mismos se administran en conjunto, con una mirada industrial y abarcativa del parque tecnológico. Se adopta un enfoque que nos permite implementar soluciones rápidas y reemplazar componentes fallidos en lugar de invertir tiempo en intentar resolver problemas complejos. Similar a como sucede en la ganadería donde el ganado se administra como un todo

¡Configuration as Code al rescate!

En los procesos modernos de configuration management, la posibilidad de definir la configuración de un servidor como código (Configuration as Code, o CaC) es fundamental.

- Gestionar el parque tecnológico como ganado no sería posible sin la utilización de un sistema de configuration management.
- Cualquier cosa definida como código puede ser automatizada.
- El código puede ser sometido a pruebas.
- CaC es un paso anterior a habilitar Self-Healing y Self-Remediation (dos prácticas del mundo moderno de infraestructura que permiten implementar procesos de autorreparación).
- Es compatible con el proceso de change management de ITIL, ya que las modificaciones no suceden en los activos, sino en un repositorio que soporta versionado. De modo que los cambios pueden ser testeados y desplegados en ambientes bajos para luego aplicarlos en producción.
- No reemplaza al proceso de configuration management de ITIL, hasta comparten el mismo nombre.

Los desafíos de adoptar Configuration as Code

Transicionar de los procesos tradicionales a los procesos modernos puede llevarnos a afrontar varios desafíos:

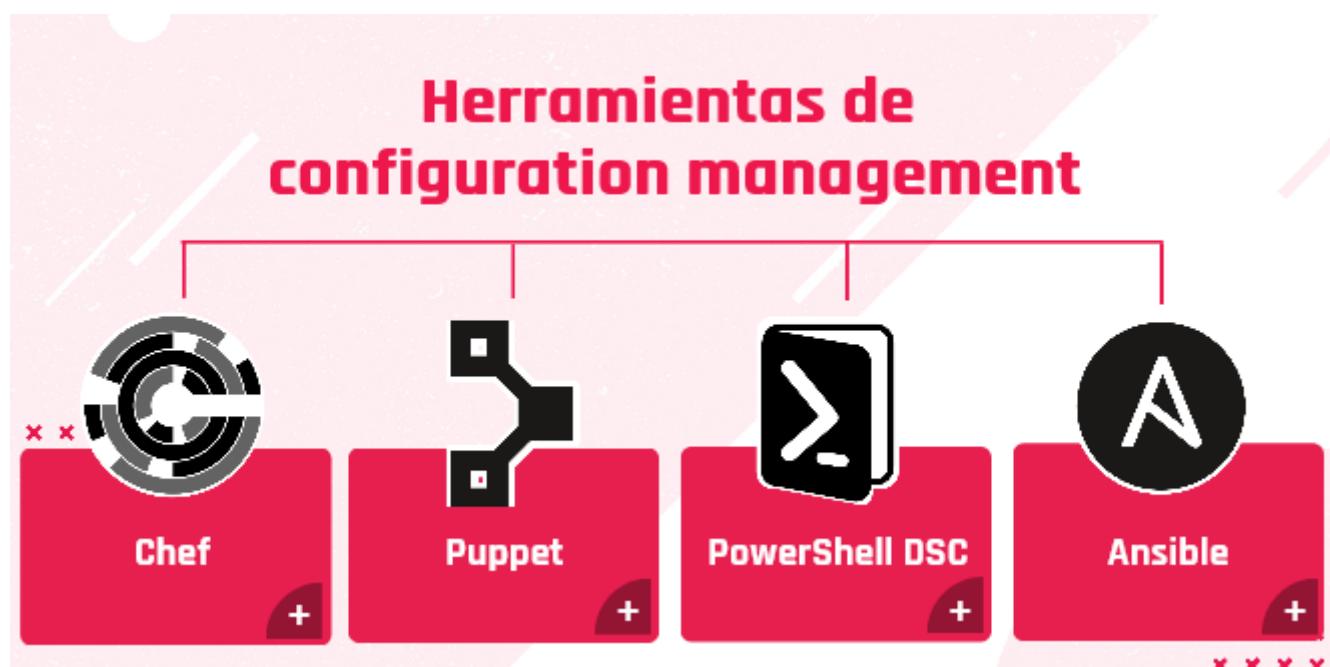
- Tenemos que entender si realmente tiene sentido transicionar a estos procesos.
- Comprender qué tan lejos queremos ir con la implementación de CaC, ¿vamos a adaptar los sistemas existentes? ¿O adoptaremos la práctica solo para los nuevos sistemas?
- Abandonar las viejas formas de trabajar puede presentar resistencia.
- Puede que necesitemos repensar la forma en la que se construyen los servidores en la organización.

Ver video: Un proceso de configuracion management moderno

Herramientas de configuration management

¿De qué manera podemos implementar la gestión de los cambios? Continuemos para averiguarlo.

Link: [Herramientas](#)



Chef

Es una plataforma de configuration management con una arquitectura cliente-servidor, principalmente, orientada a Linux. Lo que significa que las configuraciones son definidas y asignadas en el servidor y es mediante el cliente de Chef que se hacen efectivas en aquellos activos que deseamos configurar.

La presencia de un cliente hace que Chef pueda detectar las desviaciones que se producen a causa de posibles intervenciones manuales y así corregirlas para devolverla al estado deseado. El cliente de Chef funciona en modo “pull”, lo que significa que este verifica periódicamente contra el servidor si hay nuevas versiones de la configuración.

Las configuraciones de Chef se escriben en Ruby.

Puppet

Al igual que Chef, es una plataforma de configuration management con una arquitectura cliente-servidor, principalmente, orientada a Linux. Lo que significa que las configuraciones son definidas y asignadas en el servidor y es mediante el cliente de Puppet que se hacen efectivas en aquellos activos que deseamos configurar.

Como en Chef, la presencia de un cliente hace que Puppet pueda detectar las desviaciones que se producen a causa de posibles intervenciones manuales y así corregirlas. El cliente también funciona en modo “Pull”. Las configuraciones de Puppet se escriben en un DSL (domain specific language) creado por Puppet labs.

La gran diferencia entre Puppet y Chef está dada por las capacidades de reporting que tiene Puppet.

PowerShell DSC

DSC (del inglés, desired state configuration) es una tecnología que forma parte de PowerShell, originalmente desarrollada para mantener la configuración de hosts que Windows Server, es multiplataforma y puede también configurar hosts Linux.

Puede funcionar tanto en una arquitectura cliente-servidor (modo “pull”) o en una modalidad standalone (modo “push”). Cuando DSC funciona en modo “push” las configuraciones son definidas localmente y el cliente (llamado LCM, del inglés local configuration manager) es el encargado de aplicarlas sin buscar en un servidor. Tanto en modo “push” como en modo “pull”, tiene la capacidad de determinar si han habido desviaciones respecto de la configuración definida y corregirlas.

Las configuraciones para DSC son escritas en el lenguaje de scripting PowerShell.

Ansible

Es una tecnología de configuration management desarrollada en Python. Ansible no requiere de la instalación de un agente en aquellos servidores que se configuraran usando Ansible, sino que la herramienta establece una conexión vía SSH y de esa manera despliega las configuraciones.

Su versatilidad ha hecho que su uso evolucione más allá de ser utilizada como herramienta de configuration management para configurar hosts. Sino que también se utiliza como orquestador, para controlar despliegues en la nube y hasta desplegar aplicaciones como parte de un proceso de liberación de software.

Las configuraciones en Ansible se escriben en archivos con formato YAML (un superset del formato JSON).

Notas clase 10

ITIL es un conjunto de reglas y metodologias, hace referencia a buenas practicas para aplicar a infraestructura

En [Axelos](#) Se pueden hacer certificaciones ITIL, es para personas.

La norma ISO20000 esta basado en ITIL y es la que certifica empresas.

change management y configuration management, dependen una de la otra, se usan en conjunto. Se encargan de gestionar los cambios que hagamos en la infraestructura, pero se diferencian en:

Change management que gestiona esos cambios

Configuration management nos da un proceso de aprobacion para esos cambios y los va a registrar en la **CMDB - configuration management database** dando un control de los cambios que se hicieron.

Si no esta registrado, no hay control sobre ese cambio.

Da una ventaja de solucionar problemas de una manera mas dinamica y rapida.

Aplicamos esas configuraciones en formato de codigo **Configuration as code**

C11 - Ansible

¿Qué es Ansible y para qué sirve? Se trata de un software de gestión de la configuración automática y remota, que nos permite centralizar la configuración de numerosos servidores, dispositivos de red y cloud providers de una forma sencilla y automatizada.

En esta clase, lo analizaremos con más profundidad.



[Ver pdf: Conociendo Ansible.pdf](#)

Conociendo Ansible

Introducción a Ansible

- Ansible es una herramienta open source de configuration management y de aprovisionamiento, similar a Chef, Puppet o Salt.
- Usa SSH para conectarse a los servidores y ejecutar las tareas de configuración. Ansible nos permite controlar y configurar nodos desde un servidor central.
- Lo que lo hace diferente de otras herramientas de configuration management es que Ansible utiliza la infraestructura de SSH. El proyecto se originó en 2013 y, finalmente, fue comprado por Red Hat en

2015.

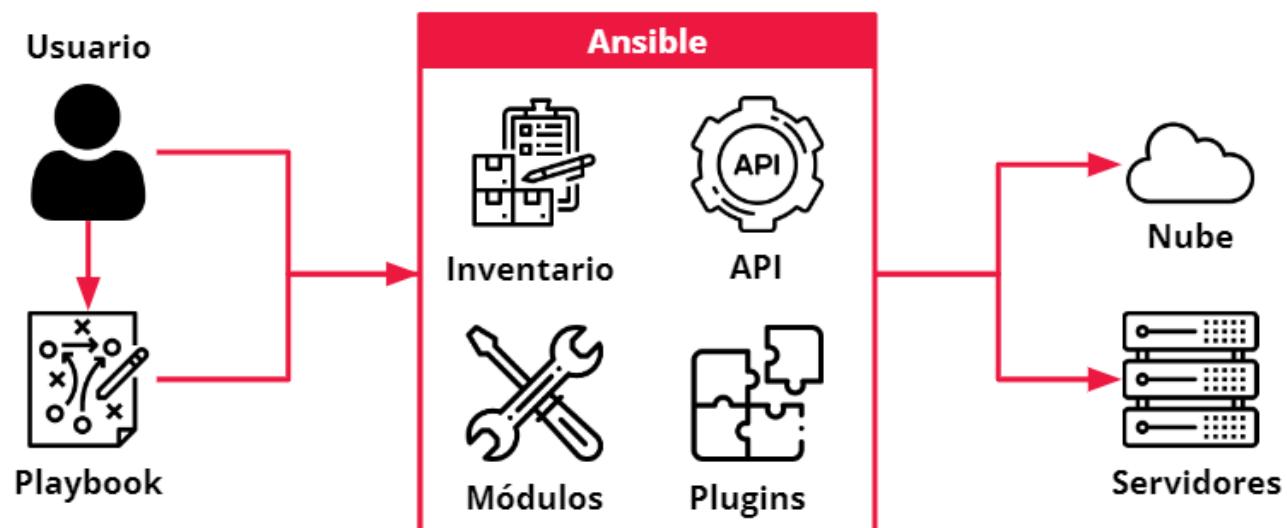
¿Por qué elegir Ansible?

- **No usa agentes:** Mientras que al equipo que queramos configurar se pueda acceder vía SSH y correr Python, se podrá configurar usando Ansible.
- **Idempotente:** Toda la arquitectura de Ansible está estructurada alrededor del concepto de idempotencia. La idea es que solo se harán configuraciones si son necesarias y que se podrán aplicar de manera repetible sin provocar efectos secundarios.
- **Declarativo:** A diferencia de un script, en donde debemos escribir la lógica necesaria para efectuar una configuración, Ansible trabaja por nosotros, dejándonos escribir una descripción del estado que deseamos para un servidor o conjunto de servidores. Es luego Ansible el que se encarga de aplicar dicha descripción de forma idempotente.
- **Fácil de aprender.**

¿Idempotencia?

- Wikipedia define a la idempotencia como: "es la propiedad para realizar una acción determinada varias veces y aun así conseguir el mismo resultado que se obtendría si se realizase una sola vez".
- En los procesos de infraestructura modernos, en donde las configuraciones se definen en **forma de código** y muchas veces de forma **declarativa**, la idempotencia no solo es una herramienta sino una necesidad para poder implementar procesos de alta predictibilidad

Arquitectura



Inventario

- El inventario es una lista de los nodos que pueden ser accedidos por Ansible. Por defecto, el inventario está soportado por un archivo de configuration, cuya ubicación es [/etc/ansible/hosts](#). Los nodos pueden estar listados por nombre o IP.
- Cada nodo es asignado a un grupo, como pueden ser "web servers", "db servers", entre otros.
- El inventario debe estar escrito en uno de muchos formatos, estos pueden ser YAML, INI, etcétera. YAML es el formato más utilizado en la industria

Ejemplo de un inventario

```
mail.example.com

[webservers]
foo.example.com
bar.example.com

[dbservers]
one.example.com
two.example.com
three.example.com
```

Playbooks

- Los playbooks son archivos también escritos en formato YAML. Estos archivos son la descripción del estado deseado de los sistemas que vamos a configurar. Ansible es el que hace todo el trabajo para llevar los servidores al estado que nosotros hayamos especificado sin importar el estado en el que estén cuando la configuración se aplique. Los playbooks hacen que las nuevas instalaciones, actualizaciones y la administración del día a día sea repetible, predecible y confiable
- Los playbooks son simples de escribir y mantener. Se escriben en un lenguaje natural por lo que son muy sencillos de evolucionar y editar.
- Los playbooks contienen Plays (jugadas).
- Las jugadas (contienen tareas (en inglés, tasks).
- Las tareas invocan módulos.

Ejemplo de un playbook

Este playbook instala la versión más reciente de Apache y se asegura que este corriendo en aquellos servidores que estén bajo el grupo "webservers" en el inventario:

```
---
- hosts: webservers
  remote_user: root

  tasks:
    - name: Asegurarse que la ultima version de Apache este instalada
      yum:
        name: httpd
        state: latest

    - name: Asegurarse que Apache este correindo
      service:
        name: httpd
        state: started
        enabled: yes
```

Módulos

- Hay más de 1000 módulos incluidos con Ansible para automatizar las diferentes partes de nuestro ambiente. Se puede pensar en los módulos como los plugins que hacen el trabajo real de configuración. Cuando se ejecutan las tareas escritas en un playbook, lo que se está ejecutando es en realidad un módulo.
- Cada módulo es independiente (no debería tener dependencia de otros módulos) y se lo puede escribir en cualquiera de los lenguajes de scripting standard de mercado (Python, Perl, Ruby, Bash, etc.). Uno de los principios de diseño de los módulos es la idempotencia.
- Dentro de los módulos más populares podemos encontrar: Service, file, copy, iptables, entre otros

Ejemplo de la invocación de un módulo

Ya vimos que los módulos se incluyen en los playbooks para componer configuraciones complejas o abarcativas. Pero también es posible invocar módulos individualmente desde la línea de comandos una única vez. Ya sea para probar el módulo o realizar una tarea específica. A continuación, vemos dos comandos, el primero reproduce una de las tareas que ya vimos en un playbook que nos ayuda a asegurarnos que el servicio de Apache está corriendo. Mientras que el segundo invoca el módulo 'ping' para hacer un 'ping' localmente contra 'localhost':

```
ansible 127.0.0.1 -m service -a "name=httpd state=started"
ansible localhost -m ping
```

La herencia de Python

Ansible está desarrollado en Python y, en consecuencia, hereda y/o implementa algunos aspectos del lenguaje. Y si bien, no es necesario ser desarrollador de Python para poder hacer uso de Ansible. Hay algunas cuestiones que es importante conocer:

- El lenguaje de templating (Jinja2)
- Operador ternario: El operador ternario de Python se puede utilizar dentro de los templates de Jinja para alterar algunos comportamientos de nuestros playbooks en función de ciertas condiciones.
- Errores: Muchas veces los errores que encontramos van a estar en un formato que, de estar familiarizados con Python, nos resultará más sencillo de leer.

Los casos de uso de Ansible

- **Aprovisionamiento:** Utilizar Ansible para instanciar servidores o máquinas virtuales “configurando el sistemas de virtualización”.
- **Configuration management:** gestionar y mantener las configuraciones de nuestros servidores.
- **App deployment:** distribuir aplicaciones.
- **Continuous delivery:** Utilizarlo como componente de un proceso de CI/CD para automatizar el despliegue de una aplicación luego de su proceso de compilación.
- **Seguridad y compliance:** La naturaleza idempotente de Ansible hace que podamos utilizarlo para distribuir configuraciones asociadas con la seguridad sin importar la configuración actual.
- **Orchestration:** Puede ser utilizado también para orquestar operaciones en la nube o ‘configurar’ la nube.

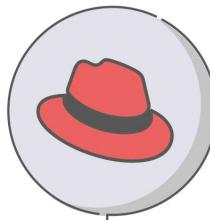
Configuration management con Ansible

Ansible es la herramientas más simple para implementar una estrategia de configuration management. Está diseñado para ser minimalista, consistente, seguro y altamente confiable. Cualquier desarrollador, tester o administrador de infraestructura puede fácilmente utilizarlo para configurar un conjunto de nodos. Además, cualquier persona en el departamento de IT podría escribir un playbook sin mayores dificultades. Las configuraciones descritas en Ansible (playbooks) son sencillamente una descripción de la infraestructura (en un lenguaje fácilmente inteligible para el ojo humano) de modo que cualquier persona en el área de IT podría entender el significado de cada tarea en un playbook. Ansible solo requiere el password o la clave privada del usuario que se utilizará para acceder desde Ansible a los sistemas que se configuraron.



¿Sabías que...?

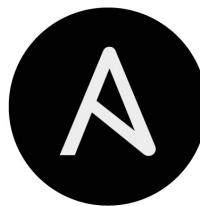
ANSIBLE



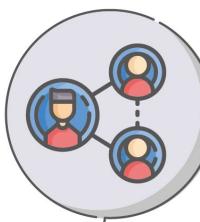
Es una herramienta mantenida por Red Hat, una empresa multinacional dedicada a la distribución de software de código abierto. Su producto más conocido es el sistema operativo Red Hat Enterprise Linux.



Tiene centenares de módulos que resuelven problemas comunes y son muy útiles. Por ejemplo, la automatización de infraestructura en la nube. La lista de módulos se encuentra en su documentación.



Existe Ansible Galaxy (<https://galaxy.ansible.com/>), una comunidad que permite la distribución de módulos hechos por los usuarios. ¡Más adelante vas a poder sumar los tuyos!



Usar Jinja2 dentro de Ansible nos permite realizar templates de scripts de powershell y bash scripting usando técnicas de programación existentes en Python.



Podemos ejecutar "roles" dentro de nuestros playbooks para crear minimódulos con código que utilizamos habitualmente.





¿Qué tipo de analista de infraestructura sos si usás Ansible?

Administrás servidores.

Ansible surge primero para gestionar configuraciones en servidores y luego se utiliza para gestionar infraestructura en la nube. Si la elegís como herramienta, es probable que también te ocupes mucho de gestionar los servidores.

Utilizás Ansible para todo.

Al ser una herramienta muy versátil, si la sabés manejar bien, podés sacarle el jugo a todas sus posibilidades.



Utilizás YAML. Ansible utiliza YAML como lenguaje para sus playbooks.

Te gusta el software open source.

El gran motivo por el que Ansible es lo que es hoy en día es por su comunidad open source. Si elegís Ansible, probablemente te guste el software open source y las comunidades que se forman alrededor de estos proyectos.

Pensás de forma descriptiva.

Te gusta describir el estado final de la infraestructura y que tu herramienta se ocupe de los pasos necesarios para llegar ahí.



Ansible en acción

¡Vamos a poner manos a la obra! Veamos cómo nos ayuda Ansible en la infraestructura IT.

[Ver pdf: Ansible en accion](#)

[Ver pdf: Configuración de IP en MV.pdf](#)

Configuración de IP en MV

Cambiar o añadir una dirección IPv4 en un servidor dedicado Linux

Por defecto, los servidores dedicados están configurados para llevar a cabo una configuración automática de la red a través de DHCP. Al servidor se le asigna siempre la primera dirección IP establecida en el contrato. Si desea cambiar la dirección del servidor o asignar una adicional, se requiere una configuración IP estática. Por razones de seguridad de la red, todos los servidores están configurados con una red /32 (máscara de red 255.255.255.255) y, por lo tanto, solo pueden llegar directamente a su puerta de enlace (gateway) y a las direcciones IP configuradas localmente. Para poder "comunicarse" con otros servidores en el segmento de red, todos los demás paquetes deben enrutararse utilizando la puerta de enlace estándar. Esto requiere una ruta de host a la puerta de enlace.

Editor Vim

El editor Vim dispone de un modo de línea o inserción y un modo de mando o comando. Podemos acceder al modo de inserción con la tecla "i". En este modo, los caracteres introducidos se insertarán inmediatamente en el texto. Para acceder al modo de comando, usamos "ESC". En este modo, todo lo que se introduzca con el teclado se interpretará como un comando. Para salir de Vim y guardar el archivo, debemos usar el comando :wq y luego presionar "Enter"

Notas de experiencia y buenas prácticas

Siempre que vamos a modificar un archivo de configuración, es de muy buena práctica hacer un backup del archivo que vamos a modificar en caso de que necesitemos volver al estado original. En este caso, vamos a modificar el archivo de configuración de la red. Por lo tanto, el primer paso será hacer un backup de la configuración:

```
usuario@debian: $ cd /etc/network/
usuario@debian:/etc/network/$ sudo cp interfaces interfaces.old
usuario@debian:/etc/network/$ ls
if-down.d if-post-down.d if-pre-up.d interfacesinterfaces.old    interfaces.d if-
up.
```

De esta manera, podremos —en caso de equivocarnos— volver al estado inicial con el comando inverso:

```
sudo cp -f intinterfaces.old interfaces
```

Debian o Ubuntu

1. Abrí el archivo `/etc/network/ifcfg-eth0` con el editor Vim.
2. Copiá y pegá el siguiente comando en `/etc/network/interfaces`, sustituyendo `<DIRECCIÓN_IPv4_PRINCIPAL>` con la dirección IP de tu servidor.

```
auto eth0
iface eth0 inet static
address <DIRECCIÓN_IPv4_PRINCIPAL>
netmask 255.255.255.255
gateway 192.168.9.1
pointopoint 192.168.9.1
```

Comprobá que la máscara de red, la puerta de enlace y la conexión Point-To-Point se hayan copiado exactamente como se especificó anteriormente. Prestá especial atención al parámetro pointopoint

El archivo de configuración editado tendrá el siguiente aspecto:

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet static
```

```
address 192.168.9.5
netmask 255.255.255.255
gateway 192.168.9.1
pointopoint 192.168.9.1
```

Por ejemplo

```
# /etc/network/interfaces -- configuration file for ifup(8), ifdown(8)
# Generated by debian-installer.
# The loopback interface
auto eth0
iface eth0 inet static
address 192.168.9.5
netmask 255.255.255.255
gateway 192.168.9.1
pointopoint 192.168.9.1
allow-hotplug eth0
iface eth0 inet6 static
address 1234:01D1:1234:B000:0000:0000:0001:E123
netmask 64
post-up ip -6 route add fe80::1 dev eth0
post-up ip -6 route add default via fe80::1 dev eth0
post-down ip -6 route del default via fe80::1 dev eth0
post-down ip -6 route del fe80::1 dev eth0
```

3. Luego, reiniciá la red con el siguiente comando:/etc/init.d/systemctl restart networking

```
/etc/init.d/systemctl restart networking
```

CentOS

1. En CentOS, abrí el archivo [/etc/sysconfig/network-scripts/ifcfg-eth0](#) con el editor Vim.
2. Aplicá los siguientes cambios para la configuración de IP estática y sustituí [`<DIRECIÓN_IPv4_PRINCIPAL>`](#)con tu dirección IP.

```
DEVICE=eth0
BOOTPROTO=static
NM_CONTROLLED="yes"
ONBOOT=yes
IPADDR=<DIRECIÓN_IPv4_PRINCIPAL>
NETMASK=255.255.255.255
```

3. Ahora, configurá la ruta necesaria. Creá un archivo de configuración [/etc/sysconfig/network-scripts/route-eth0](#) insertando el siguiente contenido:

```
ADDRESS0=192.168.9.5  
NETMASK0=255.255.255.255  
ADDRESS1=0.0.0.0  
NETMASK1=0.0.0.0  
GATEWAY1=192.168.9.
```

4. Reiniciá la red con el siguiente comando:

```
sudo service network restart
```

En la próxima clase tendrás una evaluación. Te sugerimos que vuelvas a revisar todo el material asincrónico y las ejercitaciones realizadas en las mesas de trabajo.

Dentro del examen los temas que se van a evaluar son:

Módulo 1: Inmersión
El centro de cómputos
Arquitectura cliente-servidor
Módulo 2: Automatización
Shell scripting
PowerShell
Python
Configuration management
Ansible