

Maestría en Desarrollo y Operaciones de
Software

Administración de Sistemas para la Cloud

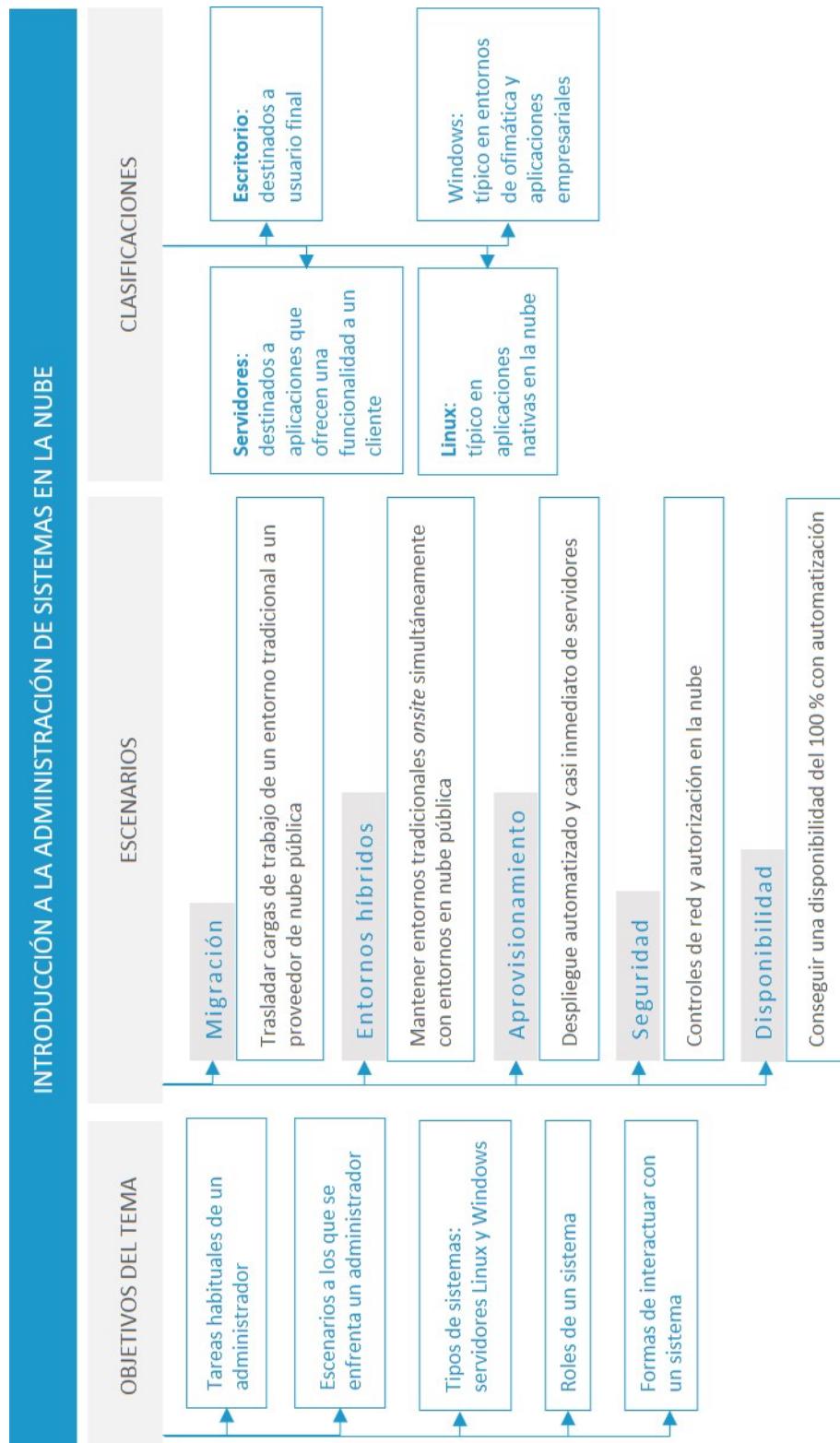
Administración de Sistemas de la Cloud

Introducción

Índice

Esquema	3
Ideas clave	4
1.1. Introducción y objetivos	4
1.2. Escenarios: migración	5
1.3. Escenarios: entornos híbridos	6
1.4. Escenarios: aprovisionamiento	6
1.5. Escenarios: seguridad	7
1.6. Escenarios: disponibilidad	7
1.7. Servidores <i>vs.</i> equipos de escritorio	8
1.8. Linux <i>vs.</i> Windows	9
1.9. Tareas tradicionales	10
1.10. Tareas en la nube	15
1.11. Referencias bibliográficas	16

Esquema



1.1. Introducción y objetivos

En este tema, se presenta el concepto de administración de sistemas en el contexto de sistemas operativos y se plantean las tareas que se esperan de un administrador de sistemas. También se presentarán otros sistemas.

Debido a la aparición de la computación en la nube, el trabajo de un administrador de sistemas está cambiando y está derivando, en muchos casos, en la reconversión hacia los **DevOps**, *development operations*, (Both, 2018). A medida que crece la adopción de la nube, la atención cambia de la administración de recursos físicos a la administración de sistemas virtuales y del mantenimiento de las máquinas a ser capaz de replicar los entornos para garantizar el servicio.

Las empresas buscarán a los administradores de sistemas y DevOps con la capacidad de liderar la transformación y el despliegue de **entornos en la nube**. Nos centraremos en los elementos generales que pueden resultar de utilidad para cualquier entorno y que serán demandados por un entorno de DevOps.

Los **objetivos** que se pretenden conseguir en este tema son:

- ▶ Conocer el alcance de las tareas habituales de un administrador de sistemas.
- ▶ Conocer los escenarios a los que se enfrenta un DevOps en un entorno de nube.
- ▶ Identificar los diferentes tipos de sistemas operativos.
- ▶ Diferenciar los roles que puede tener un sistema.
- ▶ Identificar las formas de interactuar con un sistema.

A continuación, en el vídeo *Introducción a Administración de Sistemas para Cloud*, se resumen los principales puntos que se desarrollarán en este tema.



Accede al vídeo

1.2. Escenarios: migración

Uno de los primeros retos de un administrador será la migración de cargas de trabajo de infraestructuras físicas y virtualizadas de un centro de datos privado, probablemente *onsite*, a un **proveedor de nube pública**. *Onsite* tiende a referirse a una localización propia de la organización, frente a una ubicación *offsite*, a la que la organización tiene acceso por red. Un entorno *offsite* tradicional podía ser un centro de datos o una parte de un centro de datos subalquilado a un proveedor. En estos casos, la organización cliente podía llegar a optar por administrar los servidores físicos y subcontratar el espacio y la electricidad.

¿Tiene sentido reconstruir todo el sistema desde cero? ¿Es posible crear una imagen de sistema existente que ofrece el 80 % más de la funcionalidad necesaria? La solución más sencilla puede ser hacer una copia del sistema, copiarla al proveedor de la nube y arrancarla en la nueva ubicación. Esta opción permite, entre otras cosas, **delegar la administración** de la infraestructura física y virtual al proveedor y mantener el funcionamiento familiar al que están acostumbrados los administradores. No obstante, esta opción no permitirá gestionar el ciclo de vida de las aplicaciones existentes.

En un entorno DevOps, en el que se tienda a la automatización, se debería apostar por **automatizar el despliegue**, al menos, de todas las aplicaciones nuevas y, en lo posible, de las existentes, para poder disfrutar de todas las ventajas de un entorno de nube.

1.3. Escenarios: entornos híbridos

La integración con los entornos existentes es casi inevitable cuando se emprende la migración de entornos tradicionales a la nube. Esto requiere el aprovisionamiento de recursos, tanto locales como de la nube, con protocolos compatibles. Un DevOps tendrá que combinar sus conocimientos de desarrollo y administración con **conocimientos de red**, para establecer las conexiones adecuadas. Estos recursos de red serán en muchos casos virtuales y, por lo tanto, también deben ser automatizados.

1.4. Escenarios: aprovisionamiento

La implantación del *cloud computing* permite, finalmente, que los DevOps dispongan de **recursos renovables, reciclables y fáciles de proveer**. Crear un nuevo servidor es tan simple como unos pocos clics en una consola web. El reemplazo de un servidor existente es sencillo, si está automatizado su despliegue y configuración. Además de la facilidad, el aprovisionamiento es mucho más rápido en la nube que en los entornos tradicionales. Si la carga de un servidor es muy alta, es suficiente con lanzar más instancias o, en el peor de los casos, lanzar un servidor nuevo de mayor tamaño, utilizando las automatizaciones.

Administrar un entorno en la nube significa más que mantener todos los sistemas funcionando. Los DevOps deben ahora **vigilar el coste de funcionamiento** del entorno. Los costes de *hardware* se han reemplazado por el pago por uso, a veces, pero no siempre, con el coste de licencia incluido.

Los contratos de soporte se han trasladado de los proveedores de *hardware* a los proveedores de infraestructura. La comprensión de este equilibrio ayudará a los DevOps a trabajar con otros para construir una solución rentable y confiable.

1.5. Escenarios: seguridad

La seguridad es una preocupación importante en la nube. Los proveedores son responsables de asegurar el acceso físico al *hardware*. Sin embargo, la infraestructura debe ser accedida de forma remota por definición. Los DevOps deben estar familiarizados con el **modelo de seguridad** de su proveedor de nube.

Los roles, grupos, usuarios y políticas son mecanismos comunes para **otorgar y restringir el acceso**. La mayoría de los proveedores de nube incorporan estas posibilidades de granularidad entre sus herramientas de administración. Por ejemplo, una directiva puede conceder permiso para que un usuario cree una instancia en una región y la deniegue en otra. Al igual que los entornos tradicionales, la seguridad debe ser una consideración inicial.

Permitir o bloquear el acceso a la red es un trabajo tradicionalmente de los administradores de red. En la nube es habitual que este trabajo recaiga en los DevOps, que aprovecharán la automatización para reducir los riesgos de seguridad de red.

1.6. Escenarios: disponibilidad

Uno de los objetivos de contar con recursos en la nube será que tengamos una **disponibilidad (*uptime*)** cercana al 100 %. Esto no es algo que pueda lograrse de forma automática por el mero hecho de utilizar la nube. Si bien es cierto que se podría pensar que al usar la nube no habrá caídas en los sistemas por fallos de *hardware*, no es menos cierto que las máquinas en la nube a veces necesitan ser movidas para tareas de mantenimiento o incluso apagadas por parte del proveedor. En este escenario, muy cercano a la realidad, la única forma de garantizar el *uptime* es contar con mecanismos que permitan desplegar flotas de servidores autogestionados.

El concepto de **flota** se puede concretar en un **grupo de autoescalado** en AWS (Amazon Web Services) o en un **despliegue** en Kubernetes, por citar algunos. Los objetivos detrás de un grupo de servidores autogestionados son:

- ▶ Si un servidor deja de funcionar, debe ser posible apagarlo sin intervención humana.
- ▶ Si la flota actual no tiene capacidad suficiente, debe ser posible añadir un servidor nuevo sin intervención humana.
- ▶ Una actualización incremental no debe implicar una caída del sistema.

Esta idea de flota, en la que los servidores aparecen y desaparecen sin intervención humana, queda reflejada en el **paradigma de las mascotas y el ganado** (*pets vs. cattle*) (Bias, 2016). Los servidores tradicionales eran tratados como mascotas: provisionar uno llevaba mucho tiempo y los administradores hacían lo posible para mantenerlo funcionando todo el tiempo posible. Los servidores modernos en la nube se parecen más al ganado: es posible tener tantos que no merece la pena preocuparse por cada uno en particular.

Una disponibilidad del 100 % no significa que los DevOps trabajen 24/7 para garantizarla. La automatización es la base sobre la que se construyen las aplicaciones de escalado y de despliegue.

1.7. Servidores vs. equipos de escritorio

Los apartados anteriores mencionaban tareas como el despliegue de servidores. En un entorno tradicional, un servidor podría referirse tanto al equipo informático físico, formado por la CPU (*central processing unit*), memoria, discos e interfaces de red, como al proceso de *software* que ofrece una funcionalidad a un cliente a través de la red. Aquí, el concepto de servidor se limita al de una instalación de un sistema operativo pensado para ofrecer ejecutar aplicaciones para clientes.

Los sistemas operativos usados como servidores no son muy diferente de los que ejecutan los ordenadores de sobremesa o los portátiles. Ambos tipos tienen un núcleo, una arquitectura de procesador habitualmente compatible (por ejemplo, x86), controladores de *hardware* y utilidades de *software*. Muchos de los ejemplos descritos pueden ejecutarse también en un entorno de escritorio sin cambio alguno.

Sin embargo, una instalación de servidor suele tener **dos características relevantes**: solo es posible interactuar con él por red y no tiene entorno gráfico. Un DevOps deberá escoger herramientas que le permitan adaptarse a ambas características. Como normal general, cuantas menos piezas tenga un sistema, menos pueden fallar. Si los servidores no tienen una pantalla conectada, ¿por qué no obviar el sistema gráfico totalmente?

La **línea de comandos**, esa pantalla en negro con texto que tan bien han vendido las películas sobre *hackers*, es el entorno habitual de trabajo de un administrador. Incluso Windows, que mantiene el concepto de interfaz gráfica en su nombre, introdujo PowerShell en 2007. La línea de comandos ofrece la posibilidad de escribir *scripts*: archivos con comandos y funciones propias de Bash o PowerShell que se pueden ejecutar en bloque. Efectivamente, facilitan la automatización.

1.8. Linux vs. Windows

La discusión entre **qué sistema operativo es mejor** ha alimentado multitud de foros, blog y artículos desde hace años (Economides y Evangelos, 2006; Casadesus-Masanell y Jordan, 2006; Newell, 2020). Los argumentos giran en torno al coste, la libertad de acceso al código fuente, la seguridad, la usabilidad, etc. Aquí se cubren ambos sistemas, porque, al fin y al cabo, son una herramienta y no un fin en sí mismo. Un argumento a favor en una situación (la usabilidad de Windows en un entorno de escritorio) puede no serlo en otra (en un entorno de servidor, la usabilidad del usuario puede pasar a un segundo plano).

La elección de uno u otro debe estar marcada por las necesidades de la **aplicación**, la **organización** y los **usuarios**. Por ejemplo, Linux es una gran opción para una aplicación nueva, diseñada con una arquitectura nativa en la nube, gracias al soporte nativo de contenedores. En el caso de una empresa con una gran base de usuarios que necesitan una *suite* ofimática y acceso a unas pocas herramientas corporativas, Windows parte con la ventaja de facilitar la administración de políticas de seguridad de manera centralizada.

En ambos ejemplos, se pueden dar razones a favor y en contra de uno y otro. Cuantas más herramientas domine un administrador, más preparado estará para poder evaluar dichas razones y tomar una decisión.

1.9. Tareas tradicionales

Cada organización define los roles y las responsabilidades de sus administradores. El ámbito de las tareas cambia de organización a organización y también cambia según evoluciona la organización, pero hay algunas tareas que cualquier administrador tiene (Kralicek, 2016).

- ▶ Instalación de servidores y clientes.
- ▶ Instalación y mantenimiento de aplicaciones.
- ▶ Creación de usuarios y grupos.
- ▶ Soporte a usuarios.
- ▶ Copias de seguridad y recuperación frente a desastres.
- ▶ Seguridad.
- ▶ Automatización de tareas.
- ▶ Instalación de periféricos como impresoras.
- ▶ Gestión de cambios.
- ▶ Configuración de equipos de red local.

Las organizaciones necesitarán más o menos tareas en función de su propio tamaño y del tamaño de sus departamentos. Además, en organizaciones suficientemente grandes, en entornos tradicionales que se acogen a [ITIL](#), es habitual que se implanten roles y procesos estándares.

Muchos grupos de soporte de IT (*information technology*) **dividen las tareas en roles**. Esto facilita que cada rol cree una base de conocimiento profunda para resolver problemas y ejecutar tareas. Una separación de tareas en roles podría ser la siguiente:

- ▶ Servicios de usuario:
 - Nivel 2 de soporte a usuario.
 - Instalación de periféricos.
 - Instalación y mantenimiento de aplicaciones.
 - Instalación de equipos de usuario.
- ▶ Administración de servidores:
 - Instalación de servidores.
 - Creación de usuarios y grupos.
 - Copias de seguridad y recuperación frente a desastres.
 - Automatización de tareas.
- ▶ Seguridad IT:
 - Soporte de red.
 - Gestión de cambios.

En este ejemplo, las tareas se separan en **tres roles**, que facilitan la separación de responsabilidades y, por tanto, las operaciones y la resolución de problemas. Esto, a su vez, incrementa las habilidades y el entrenamiento técnico del personal en cada rol. Las organizaciones más pequeñas suelen unir estos roles. No hay una división ideal para todos los tamaños de organización y cada una adaptará su estructura de roles al tamaño de su flota de equipos, a la complejidad de las tareas o a ambos.

En este caso, muchas de las tareas de los administradores deberán ser automatizadas y genéricas; es decir, con la posibilidad de reusar gran parte de los *scripts* y de usar *scripts* disponibles *online* y en libros del sector.

Una manera de optimizar las horas de un equipo pequeño es **estandarizar el despliegue de equipos**, ya sean servidores o de escritorio. Cuanto más se parecen los servidores entre sí, más fácil es resolver los problemas que puedan aparecer en el día a día. Esto, además, facilita el soporte y mejora su calidad.

Operaciones

En departamentos grandes y maduros, los procedimientos y los mecanismos están bien establecidos. Estos procesos están integrados en aplicaciones de soporte y enlazan tanto aspectos operacionales como procesos clave de la administración del servicio, como gestión de cambios, control de configuración, inventario, catálogo de servicios, gestión de incidencias, etc. Estas aplicaciones son complejas y caras y requieren un conocimiento avanzado de los procesos de negocio y de la naturaleza técnica del mismo.

En departamentos pequeños, sin embargo, el nivel de madurez de la documentación de los procesos y de la gestión del conocimiento puede no ser homogénea. En estos casos, un administrador puede usar una lista diaria o semanal para gestionar sus tareas, además de coordinarse con otros equipos para asegurarse de que todos los sistemas funcionan correctamente.

Comunicación

Es habitual no tener en cuenta que **la información debe fluir en dos sentidos**. En muchos departamentos, los administradores reciben peticiones de soporte y se ven obligados a cerrarlas rápido, penalizando la calidad del soporte, debido a las métricas con las que se los evalúa a final de año. Esto limita la involucración del usuario que

inició la comunicación, así como del resto de los individuos que han formado parte de la resolución.

Tanto la comunicación con otros equipos como la interna benefician a todas las partes. Internamente, estar al tanto de lo que ocurre o de cómo se ha solucionado un problema puede ayudar en futuros problemas. La documentación de estas soluciones se suele llevar a cabo en un *knowledge base* (KB), o **base o biblioteca de conocimiento**. Incluso, algunas organizaciones publican estos KB al exterior (bien públicamente o al menos a un sector interno más amplio que el propio equipo de IT), para facilitar la resolución proactiva de problemas.

Esto último enlaza con la necesidad de educar e informar al usuario. Si los usuarios finales reciben información sobre la resolución de sus problemas, a largo plazo facilitan el trabajo de los equipos de soporte. También ayuda al hacer que todos estén al tanto de las tareas del día a día y el mantenimiento preventivo. Un objetivo de los departamentos es tener una **política 90-8-2**:

- ▶ Los usuarios resuelven por sí mismos el 90 % de las incidencias.
- ▶ Los grupos de soporte intermedios se encargan de resolver un 8 %, que serán situaciones más complicadas, pero normalmente documentadas y en las que no hay que involucrar a un experto.
- ▶ Los administradores solo reciben un 2 % de los problemas. Así se pueden dedicar a tareas en las que añaden valor a la organización.

Investigación

Estar al tanto de las novedades en IT ayuda a ser proactivo con posibles problemas externos. Hay muchas revistas técnicas gratuitas ([Information Week](#), [Redmond Magazine](#), [Information Security Magazine](#)), así como excelentes sitios de investigación en la web ([Whatis.com](#), EventID.net, [Tech Republic](#)). Ampliar los conocimientos con este tipo de recursos mejora la sensibilidad operacional.

Formación

Asistir a **cursos específicos de proveedores** para la certificación no solo ayuda al desarrollo profesional del administrador, sino que también beneficia a la organización, al aumentar su conocimiento en la gestión de problemas. De hecho, las certificaciones hacen que los administradores y la operativa de IT sean más respetadas por sus clientes y, en algunos casos, es un requisito para optar a un contrato. Proporciona experiencia interna reconocida por la industria y asegura a quienes utilizan sus servicios que la organización cumple con los estándares. Además, facilita el *networking* y no en el sentido técnico, sino en cuanto a las relaciones profesionales con otros individuos del sector.

Confianza

Un sistema bien administrado permite que las organizaciones lleven a cabo sus negocios de manera estable, sin necesidad de que los usuarios de las aplicaciones tengan que comprender ni su arquitectura ni la operativa necesaria. Cuando se consiguen estos objetivos, se promueve la confianza de la organización y de los usuarios en el equipo de IT y viceversa.

¿Qué hay que cambiar?

Las ideas mencionadas en esta sección no son exclusivas de entornos tradicionales con centros de datos propios, servidores físicos y equipos de escritorio Windows, por citar un ejemplo lo más tradicional posible. Un equipo DevOps aplicará muchos de estos principios en su día a día: deberá organizar sus tareas, probablemente se especializarán en roles en función de sus fortalezas y, en cierta medida, estarán involucrados en tareas de soporte. Quizás no sigan ITIL a rajatabla, pero eso no impide que documenten sus procesos o estandaricen sus roles, tareas y herramientas.

Aunque las tareas técnicas cambien, los administradores de sistemas siguen teniendo el mismo objetivo común: apoyar a las organizaciones a conseguir sus objetivos de negocio.

1.10. Tareas en la nube

La función del administrador del sistema está cambiando. Hace solo unos años, la mayoría de los sistemas se ocupaban de granjas de servidores de *hardware* físico y realizaban una planificación detallada de la capacidad. Ampliar su aplicación significaba comprar un nuevo *hardware* y, tal vez, pasar tiempo acumulándolo en el centro de datos. Actualmente, hay un gran porcentaje de la industria que nunca ha tocado el *hardware* físico. Es posible desplegar servidores con una llamada a API (*application programming interfaces*) o haciendo clic en un botón en una página web (Lucifredi y Ryan, 2018).

Aunque el término ha sido apropiado por los equipos de *marketing*, la **nube** es algo sorprendente. En este contexto, se usa el término «nube» para referirse a la idea de servicios informáticos y de aplicaciones escalables a demanda, en lugar de servicios «basados» en la nube, como Google Mail.

A medida que aumenta la competencia en el espacio del mercado de la nube, su **atractivo**, para los administradores de sistemas y los propietarios de negocios, **aumenta casi a diario**. Amazon Web Services continúa impulsando el mercado de la computación en la nube al introducir con frecuencia nuevas herramientas y servicios (no hay más que repasar la velocidad con la que publican las novedades en [What's New](#)).

Las economías de escala están constantemente bajando el precio de los servicios en la nube. Aunque los entornos como AWS o Google Compute Engine aún no son adecuados para todas las aplicaciones, cada vez es más claro que las habilidades en

la nube se están convirtiendo en una **parte necesaria** de un conjunto de herramientas completo del administrador de sistemas.

Para las empresas, la nube abre nuevas **vías de flexibilidad**. Los equipos tecnológicos pueden hacer cosas que hubieran sido prohibitivamente caras hace solo unos años. Los juegos y las aplicaciones que tienen la suerte de convertirse en éxitos desbocados a menudo requieren una gran cantidad de capacidad de cómputo. Provisionar esta capacidad en horas en lugar de semanas permite a estas compañías responder rápidamente al éxito, sin entrar en inversiones y compromisos de gasto por varios años o gastos iniciales de capital.

En la era DevOps, los desarrolladores y la dirección saben lo importante que es iterar y mejorar rápidamente el código de aplicación. Los servicios de los proveedores de nube permiten tratar la infraestructura de la misma manera, permitiendo que un equipo relativamente pequeño administre infraestructuras de aplicaciones masivamente escalables.

1.11. Referencias bibliográficas

Axelos. (s. f.). *ITIL*. <https://www.axelos.com/best-practice-solutions/itil>

AWS. (s. f.). *Novedades de AWS*. https://aws.amazon.com/es/new/?whats-new-content-all.sort-by=item.additionalFields.postDateTime&whats-new-content-all.sort-order=desc&awsf.whats-new-analytics=*all&awsf.whats-new-app-integration=*all&awsf.whats-new-arvr=*all&awsf.whats-new-cost-management=*all&awsf.whats-new-blockchain=*all&awsf.whats-new-business-applications=*all&awsf.whats-new-compute=*all&awsf.whats-new-containers=*all&awsf.whats-new-customer-enablement=*all&awsf.whats-new-customer%20engagement=*all&awsf.whats-new-database=*all&awsf.whats-new-developer-tools=*all&awsf.whats-new-end-user-computing=*all&awsf.whats-new-

[mobile=*all&awsf.whats-new-gametech=*all&awsf.whats-new-iot=*all&awsf.whats-new-machine-learning=*all&awsf.whats-new-management-governance=*all&awsf.whats-new-media-services=*all&awsf.whats-new-migration-transfer=*all&awsf.whats-new-networking-content-delivery=*all&awsf.whats-new-quantum-tech=*all&awsf.whats-new-robotics=*all&awsf.whats-new-satellite=*all&awsf.whats-new-security-id-compliance=*all&awsf.whats-new-serverless=*all&awsf.whats-new-storage=*all](#)

Bias, R. (2016, septiembre 29). *The History of Pets vs Cattle and How to Use the Analogy Properly*. Cloudscaling.

<https://cloudscaling.com/blog/cloud-computing/the-history-of-pets-vs-cattle/>

Both, D. (2018). *The Linux Philosophy for SysAdmins: And Everyone Who Wants To Be One* (cap. 1, pp. 3-14). Apress.

Casadesus-Masanell, R. y Jordan, M. (2006). *Linux vs. Windows*. Harvard Business School. <https://www.hbs.edu/faculty/Pages/item.aspx?num=33719>

Economides, N. y Evangelos, K. (2006). Linux vs. Windows: A comparison of application and platform innovation incentives for open source and proprietary software platforms. En Bitzer, J. y Srhroder, P. (Eds.), *The Economics of Open Source Software Development* (pp. 207-218). Elsevier.

Kralicek, E. (2016). *The Accidental Sysadmin Handbook* (2.ª ed.). Apress.

Lucifredi, F. y Ryan, M. (2018). *AWS System Administration*. O'Reilly Media.

Newell, G. (2020, mayo 29). *12 Reasons Why Linux Is Better Than Windows 10*. Lifewire. <https://www.lifewire.com/windows-vs-linux-mint-2200609>

Página de *Information Week* (<https://www.informationweek.com/>).

Página de *Infosecurity* (<https://www.infosecurity-magazine.com/>).

Página de *Redmond* (<https://redmondmag.com/Home.aspx>).

Página de TechRepublic (<https://www.techrepublic.com/>).

Página de WhatIs.com (<https://whatis.techtarget.com/>).

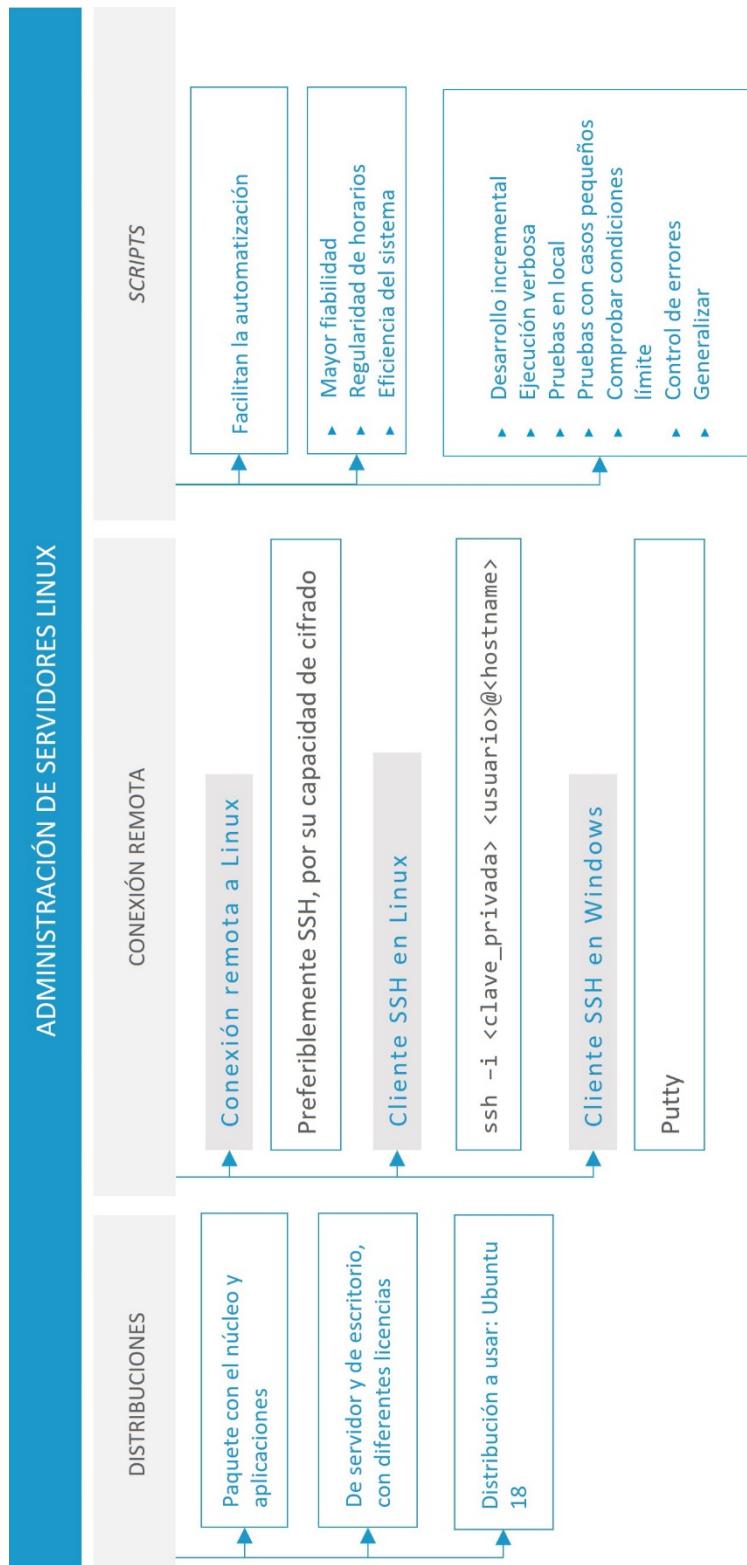
Administración de Sistemas en la Cloud

Administración de servidores Linux

Índice

Esquema	3
Ideas clave	4
2.1. Introducción y objetivos	4
2.2. Distribuciones	4
2.3. Acceso remoto	5
2.4. Servidor SSH	7
2.5. Cliente SSH	8
2.6. SSH con clave privada	14
2.7. Automatización de instalación y configuración	17
2.8. Acceso remoto en la nube	19
2.9. Referencias bibliográficas	26

Esquema



2.1. Introducción y objetivos

Este tema presentará el concepto de distribución y cubrirá en detalle el acceso remoto a Linux.

Los **objetivos** que se pretenden conseguir son:

- ▶ Entender el concepto de distribución y su relación con el sistema operativo Linux.
- ▶ Conocer las diferentes formas de acceso remoto que ofrece Linux.
- ▶ Entender los conceptos básicos de automatización e instalación.

2.2. Distribuciones

Una distribución de Linux es un paquete que consiste en el **núcleo del sistema operativo** y una **colección de paquetes y aplicaciones** (Dulaney, 2018). Dada una misma versión del núcleo, una aplicación escrita para Linux debería ser ejecutable en cualquier distribución con ese núcleo. No obstante, dado que la colección de aplicaciones abarca desde el instalador al entorno gráfico y el gestor de paquetes, la manera de interactuar con cada distribución puede ser muy diferente. La licencia también varía de una distribución a otra: aunque Linux es de código abierto, hay distribuciones comerciales que se ofrecen con contratos de soporte. Compañías como SUSE o RedHat no cobran por el sistema operativo, sino por el soporte que ofrecen a sus clientes.

Algunas distribuciones están enfocadas al usuario final. A tal efecto, incluyen entorno gráfico, aplicaciones ofimáticas, reproductores de audio y vídeo, etc. Aquí nos

centraremos en **distribuciones orientadas a servidores**. Algunas de ellas, como Ubuntu, mantienen ediciones de escritorio y de servidor para una misma versión de la distribución.

Entre las aplicaciones incluidas en casi todas las distribuciones, están las **utilidades GNU**. Hay utilidades de manipulación de ficheros y del propio contenido de los ficheros, de información del sistema operativo, de la sesión, de búsqueda, editores de texto, etc. Estas utilidades son esenciales a la hora de administrar servidores en la línea de comandos y para automatizar tareas.

Los ejemplos y ejercicios que utilizaremos asumirán que se usa una instalación de [Ubuntu 18.04](#), salvo que se indique lo contrario. Es una distribución basada en [Debian](#), que a su vez es una de las distribuciones más utilizadas como base para otras. Ubuntu es muy habitual en entornos de escritorio y relativamente habitual en entornos de servidor. Otras distribuciones muy habituales en entornos corporativos son SUSE, [Red Hat Enterprise Linux](#) (RHEL) y su variante gratuita [CentOS](#). AWS (Amazon Web Services) ofrece [Amazon Linux 2](#) como imagen base para instancias Linux, así como imágenes para VirtualBox y otros entornos de virtualización para pruebas en local.

2.3. Acceso remoto

Uno de los primeros aspectos a considerar para utilizar un servidor en la nube es el acceso remoto al mismo. Al tratarse de un servidor en la nube, será necesario contar con las **herramientas necesarias** que lo permitan:

- ▶ Telnet.
- ▶ SSH (Secure Shell).
- ▶ Otros.

Telnet

Telnet es un protocolo de capa de aplicación, utilizado en Internet o redes de área local, para proporcionar acceso remoto mediante una conexión de terminal virtual. La utilización de Telnet está desaconsejada, porque el tráfico no se transmite cifrado. Este hecho permitiría a un usuario malintencionado acceder al contenido del tráfico, si fuera capaz de interceptarlo. Para poder utilizar Telnet, es necesario que el servidor lo tenga activado y contar con un cliente de Telnet, que habitualmente está incluido en todos los sistemas operativos.

SSH

Secure Shell (SSH) es un protocolo de red de cifrado para servicios de red de operación segura a través de una red no segura. Su aplicación más habitual es la de inicio de sesión remoto a servidores Linux. SSH proporciona un canal seguro mediante una arquitectura cliente-servidor. Los sistemas operativos Linux cuentan habitualmente con un cliente de SSH. En el caso de los clientes Windows, el cliente más utilizado es el llamado [Putty](#). Se trata de un cliente de SSH de código abierto y libre (Van Vugt, 2015).

SSH es la forma más habitual de acceder a los servidores, tanto aquellos alojados en la nube como servidores tradicionales en un centro de datos propio. SSH soporta el acceso con usuario y *password* y el uso de certificados (también llamados pareja de claves o *key pair*). En muchos de los proveedores de nube pública, el uso de certificados es obligatorio.

Otros

Existen otras formas de acceder a los servidores de forma remota, la mayoría de ellas específicas para entornos gráficos, por lo que serán menos usadas a nivel de automatización y administración. **Algunas** de ellas son:

- ▶ **VNC.** Virtual Network Computing (VNC) es un sistema de control remoto que utiliza el protocolo *remote framebuffer* (RFB) para controlar de forma remota otro ordenador. Transmite los eventos de teclado y ratón de un ordenador a otro y las actualizaciones de pantalla de vuelta en la otra dirección. VNC es independiente de la plataforma, por lo que puede usarse para conectarse también a equipos Windows. El código fuente VNC original y muchos derivados modernos son de código abierto, bajo la licencia pública general, GNU por sus siglas en inglés.
- ▶ **TeamViewer.** Se trata de un desarrollo comercial y propietario, pero es posible usarlo de forma gratuita en una de sus ediciones. Puede ser útil en ciertos casos para acceder a equipos de sobremesa para soporte remoto o como substituto de una VPN, pero rara vez se usa para tareas de administración.
- ▶ **LogmeIn.** Se trata de un *software SaaS* que permite la gestión, acceso y monitorización de la PC (*personal computer*) o servidores de forma multiplataforma y mediante su interfaz web.

2.4. Servidor SSH

En este apartado, se detallará paso a paso **cómo conectarse a un servidor Linux mediante SSH**. El sistema operativo de una instancia en la nube estará preparado para aceptar conexiones SSH por defecto. Sin embargo, en un entorno local, es posible que sea necesario activar el servicio SSH. Por ejemplo, en una instalación de Ubuntu 18 en una máquina virtual, SSH no está instalado por defecto.

A continuación, en el vídeo *Instalación de Ubuntu y acceso remoto al servidor*, se podrá ver una demostración de instalación de Ubuntu 18 Server y cómo es el acceso al servidor.



Accede al vídeo

Para instalarlo, es necesario seguir estos **pasos**:

- ▶ Ejecutar `sudo apt-get update && sudo apt-get install -y openssh-server`.
- ▶ Asegurarse de que el servicio esté arrancado con `sudo service ssh start`.
- ▶ Editar el archivo de configuración con `sudo vim /etc/ssh/sshd_config` o con cualquier otro editor de texto. El fichero debe contener al menos las siguientes opciones. Estas opciones no son las más seguras, pero servirán para poder hacer pruebas en entornos locales:
 - `Port 22`.
 - `ListenAddress 0.0.0.0`.
 - `PasswordAuthentication yes`.
- ▶ Recargar la configuración con `sudo service sshd reload`.
- ▶ Averiguar la IP (*internet protocol*) del equipo con `ip address`.

En este punto, ya sería posible usar un cliente para conectarse y, en un entorno local, no será necesario más configuración. En un entorno de nube, será necesario haber habilitado el puerto 22 en el grupo de seguridad de la instancia.

2.5. Cliente SSH

Asumiendo que el servidor SSH ha sido configurado correctamente, se podría usar el cliente SSH desde un equipo Linux o MacOS con SSH `<nombre_usuario>@<ip_servidor>`, por ejemplo: `ssh ubuntu@192.168.1.130`.

```
1. ubuntu@ubuntu-VirtualBox: ~
~ $ssh ubuntu@192.168.1.130
ubuntu@192.168.1.130's password:
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-29-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

 * Canonical Livepatch is available for installation.
 - Reduce system reboots and improve kernel security. Activate at:
   https://ubuntu.com/livepatch

601 packages can be updated.
358 updates are security updates.

Last login: Sun Apr 19 18:14:25 2020 from 192.168.1.134
ubuntu@ubuntu-VirtualBox:~$
```

Figura 1. Inicio de sesión en Ubuntu con SSH. Fuente: elaboración propia.

Para acceder desde un equipo Windows, se podría usar **Putty**. Es el cliente SSH recomendado para entornos Windows, está disponible para su descarga y es gratuito. Tras ejecutar Putty, se presentará la primera pantalla de configuración (ver Figura 2).

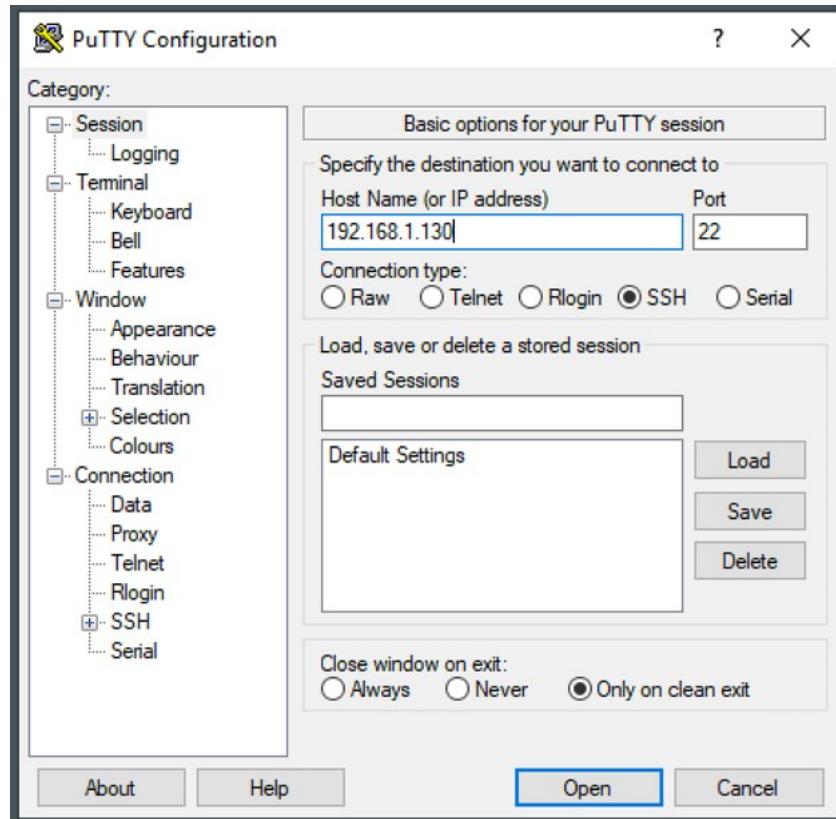


Figura 2. Pantalla de inicio de Putty. Fuente: elaboración propia.

En esta pantalla, será posible configurar la información para memorizar diferentes sesiones de conexión. En el caso más sencillo, será necesario marcar *connection type* a **SSH** e introducir la IP en el campo **Host Name (o IP address)**. Tras presionar el botón Open, se abrirá una ventana de terminal donde se solicitará el usuario y la contraseña para intentar el inicio de sesión. En el primer intento de conexión, aparecerá una advertencia que el equipo es desconocido. Una vez aceptado este aviso, no volverá a aparecer, a menos que se use el mismo nombre de equipo o la misma IP para conectarse a un equipo diferente.

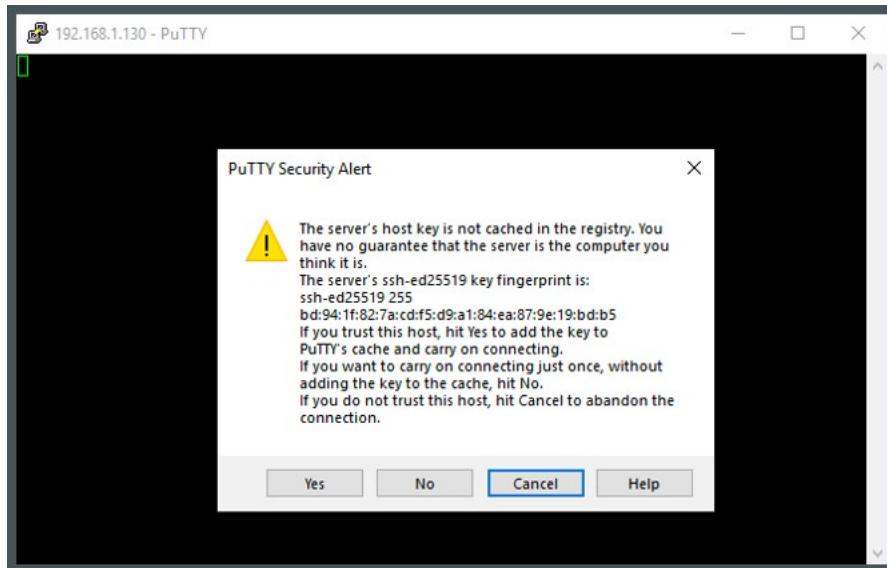


Figura 3. Advertencia en Putty durante la primera conexión a un equipo. Fuente: elaboración propia.

Una vez introducido el usuario y la *password* correcta, Putty ofrecerá una consola en la máquina remota, como muestra la Figura 4.

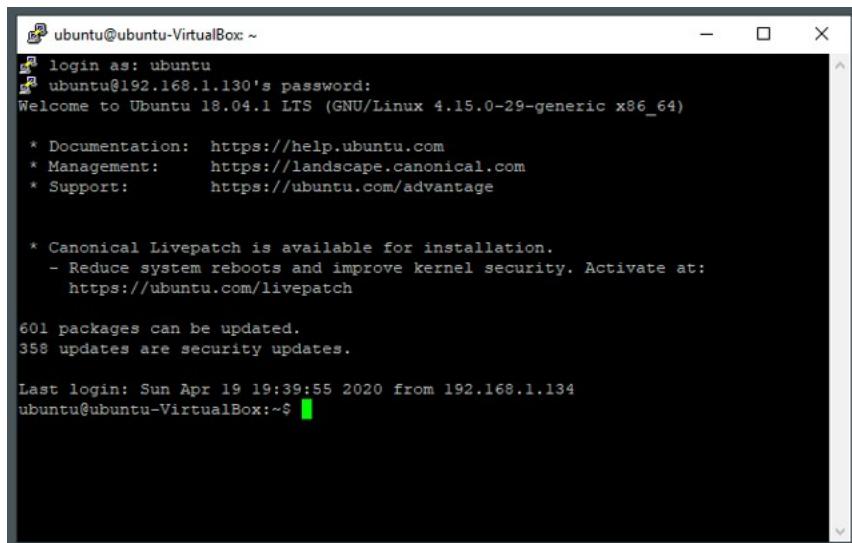


Figura 4. Sesión remota en Ubuntu con Putty. Fuente: elaboración propia.

Putty dispone de más opciones de configuración, accesibles desde el ícono de la ventana (ver Figura 5).

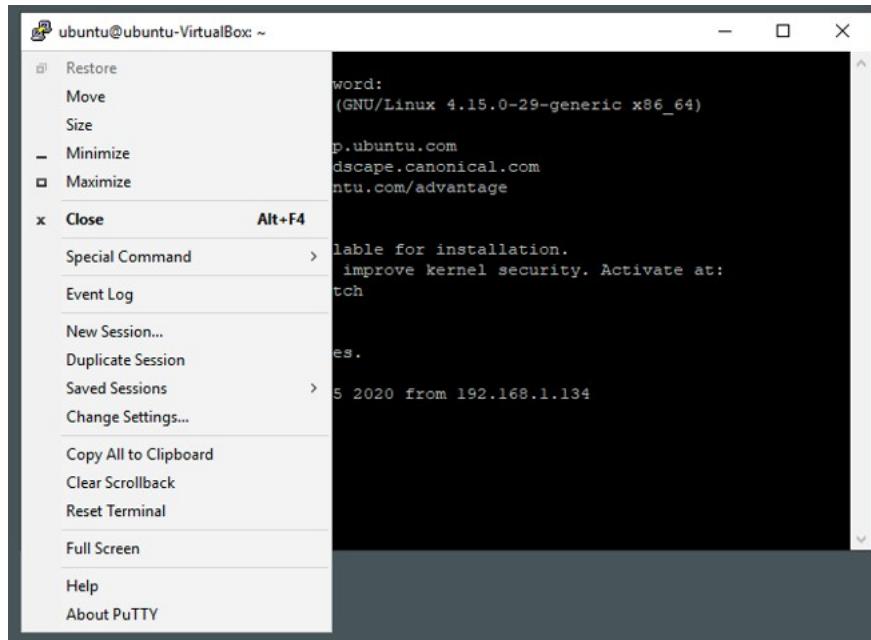


Figura 5. Menú contextual de Putty. Fuente: elaboración propia.

- ▶ **Special Command:** muestra una lista de los comandos más frecuentes en esa máquina, que pueden ser enviados directamente por Putty como una forma de ayudar al usuario a realizar tareas de forma más rápida.
- ▶ **Event Log:** registro de todas las operaciones realizadas en esa máquina durante la sesión de conexión. Esta opción resulta especialmente útil en procesos de automatización, donde será habitual el probar manualmente los comandos que instalan o configuran un componente *software*. Este registro puede usarse como primer borrador del *script*.
- ▶ **New Session:** permite realizar una nueva conexión a este u otro servidor diferente.
- ▶ **Duplicate Session:** permite abrir de forma automática una nueva sesión contra la misma máquina.
- ▶ **Saved Sessions:** permite acceder a las conexiones guardadas.
- ▶ **Change Settings:** esta opción da acceso al cambio de parámetros de la conexión actual.

- ▶ **Copy All to Clipboard:** esta funcionalidad copia todo el contenido del Buffer de la sesión actual al portapapeles. Esta opción es muy útil a la hora de generar *scripts* a partir de los resultados de una sesión interactiva.
- ▶ **Clear Scrollback:** limpia el contenido de la pantalla de texto.
- ▶ **Reset Terminal:** reinicia el terminal de texto, borrando la pantalla y limpiando el buffer.
- ▶ **Full Screen:** ejecuta Putty en modo pantalla completa.

Putty permite guardar los comandos tecleados en la consola y la salida de estos en un archivo de *log*. El fichero de salida se selecciona en la ventana de configuración.

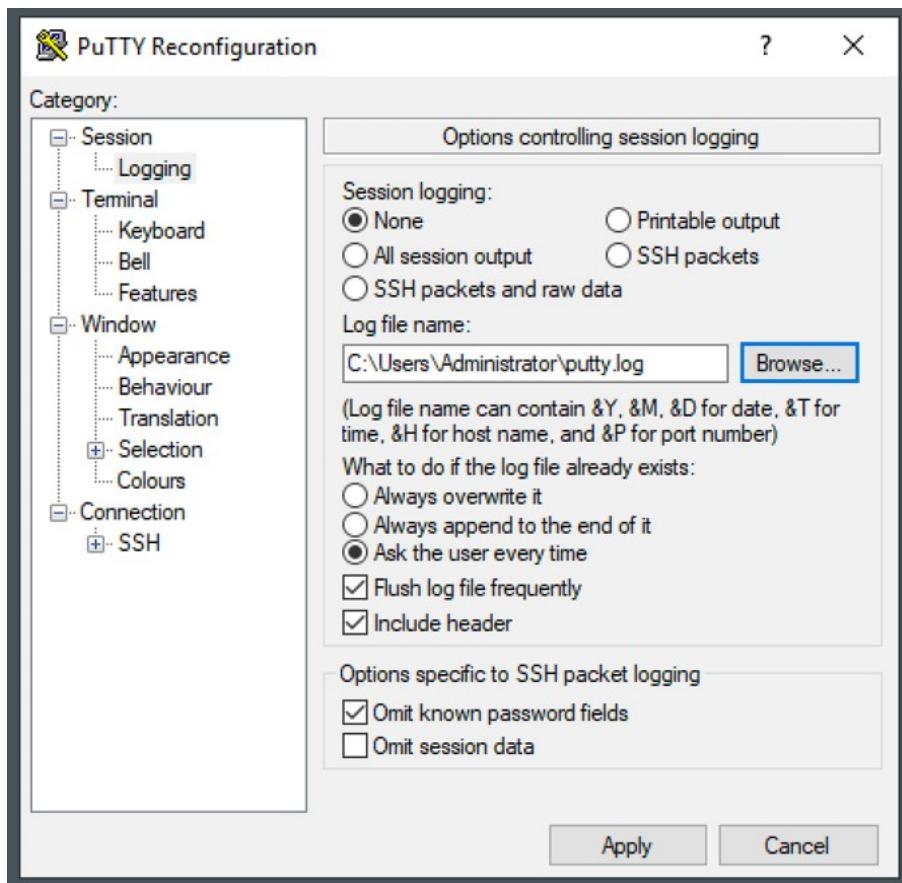


Figura 6. Configuración de *log* de Putty. Fuente: elaboración propia.

2.6. SSH con clave privada

Los ejemplos anteriores son útiles para pruebas locales, pero, en escenarios reales, es más seguro **no utilizar contraseñas para los inicios de sesión**. SSH soporta autenticación basada en claves: el usuario crea una pareja de claves, pública y privada, y copia la clave pública a aquellos servidores a los que se va a conectar. Al conectar, selecciona la clave privada en el cliente SSH. El servidor, al recibir la conexión, comprobará que la clave privada corresponde con la clave pública (Matotek et al., 2017).

Los proveedores de nube suelen ofrecer la opción de generar una **pareja de claves** y copiar la clave pública en los servidores. El usuario deberá guardar la clave privada y proporcionarla al cliente de SSH en cada conexión. Por ejemplo, si la clave privada está alojada en key.pem y la instancia ha sido desplegada en AWS con la imagen base de Amazon Linux, el comando sería parecido a:

```
ssh -i key.pem ec2-user@34.123.123.123
```

Es posible generar la pareja de **claves localmente** y copiar la clave pública a los servidores por algún otro método o proporcionarla al proveedor de nube antes de arrancar las instancias. En una máquina Linux, se podría generar la pareja de claves de la siguiente manera (Matotek et al., 2017):

```
ssh-keygen -t rsa -b 4096 -f ./security_key
```

La clave pública se guardará en el fichero security_key.pub y la clave privada en security_key. Por defecto, las rutas serían ~/.ssh/id_rsa.pub y ~/.ssh/id_rsa. Además, el comando solicita una contraseña (o *passphrase*) con la que proteger el fichero de clave privada. A continuación, es necesario copiar la clave pública al equipo remoto con el siguiente comando:

```
ssh-copy-id -i ./security_key.pub <usuario>@<equipo_remoto>
```

A partir de ese momento, ya es posible conectarse indicando el fichero de clave privada. El cliente solicitará la clave del fichero, pero no hace falta indicar la contraseña del usuario:

```
ssh -i ./security_key <usuario>@<equipo_remoto>
```

También es posible generar y usar claves públicas y privadas en Windows con **Puttygen**, una utilidad disponible en la misma web que Putty. Haciendo *click* en Generate, se obtienen las claves que habrá que guardar en dos ficheros con Save public key y Save private key (ver Figura 7).

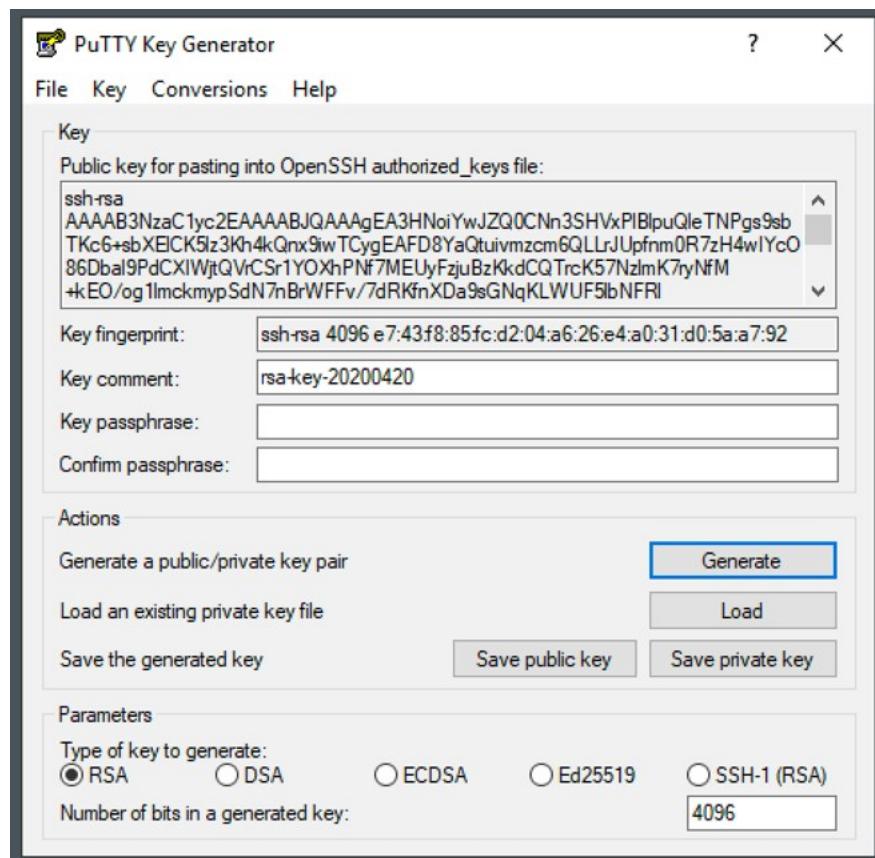


Figura 7. Interfaz de Puttygen. Fuente: elaboración propia.

Para copiar la clave pública, es necesario conectarse al equipo con contraseña, abrir el fichero `~/.ssh/authorized_keys` y pegar en una línea nueva el contenido de la ventana de Puttygen que empieza por `ssh-rsa`. Por ejemplo, usando el editor nano, el fichero quedaría como muestra la Figura 8.

```

ubuntu@ubuntu-VirtualBox: ~/ssh
GNU nano 2.9.3           authorized_keys

ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQCK0K5gY5AAPjHjEtxzraUedHg83YNP12BDyZc3QMP$  

ssh-rsa AAAAB3NzaC1yc2EAAAABJQAAgEA3HNoiYwJZQOCNn3SHVxP1BlpuQleTNPgs9sbTKc6+sb$
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
 ^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line

Figura 8. Edición de authorized_keys con nano. Fuente: elaboración propia.

La primera línea es la de la clave pública generada en el equipo Linux y que el comando ssh-copy-id se encargó de copiar automáticamente. La segunda ha sido copiada a mano desde Puttygen. El último paso es configurar Putty para usar el fichero de clave privada guardado anteriormente. Se especifica en **Connection > SSH > Auth > Private key file for authentication**.

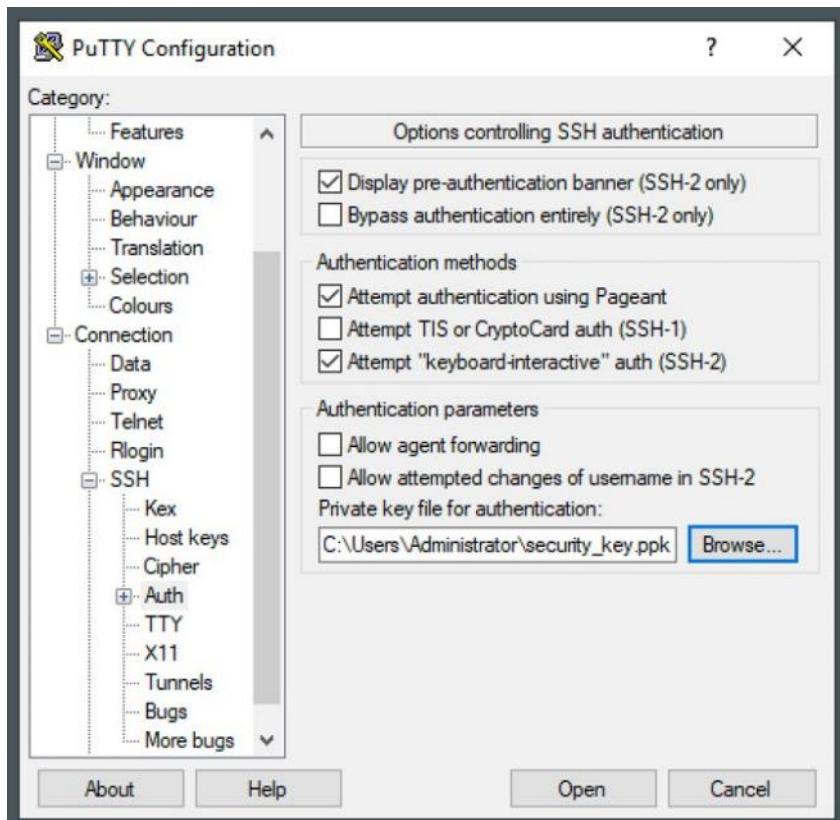


Figura 9. Selección de clave privada en Putty. Fuente: elaboración propia.

Los pasos que se han detallado hasta ahora pueden parecer tediosos, pero son fácilmente automatizables, bien en la línea de comandos o bien con herramientas de despliegue como Ansible.

2.7. Automatización de instalación y configuración

La automatización es fundamental para un administrador de sistemas. Aunque el perfil de un administrador no es el mismo que el de un programador, la capacidad de escribir *scripts* para **automatizar tareas** es una gran herramienta en el arsenal del *sysadmin* (administrador de sistemas).

Un *script* será una pieza de código que implementa una tarea concreta. Un *script* útil para un administrador estará escrito en un lenguaje interpretado y que esté disponible en el sistema de destino. Aquí se tratarán principalmente *scripts* en Bash, ya que esta *shell* está disponible en muchas distribuciones Linux, pero un *script* puede estar escrito en Perl, en Python o en cualquier otro lenguaje, siempre que el administrador esté cómodo con él.

En general, la automatización mediante *scripts* presenta las siguientes **ventajas** sobre la ejecución manual (Frisch, 2002):

- ▶ Mayor fiabilidad: una vez se ha comprobado que un *script* funciona bien, funcionará de la misma manera todas las veces.
- ▶ Regularidad de horarios: las tareas automatizadas pueden programarse en base a un horario concreto o a eventos externos; es decir, no es necesaria la presencia del administrador.
- ▶ Eficiencia del sistema: las tareas administrativas intensivas se pueden ejecutar en horarios de mejor carga del sistema.

A continuación, se indican algunas **estrategias** para escribir *scripts* útiles (Frisch, 2002):

- ▶ **Escribir el *script* incrementalmente.** Conviene escribir una versión sencilla del *script* e ir añadiendo funcionalidad poco a poco. Por ejemplo, la primera versión puede no aceptar argumentos en invocación del *script* para no añadir ruido al código de la funcionalidad que se quiere implementar.
- ▶ **Invocar los *scripts* en modo verboso.** Por ejemplo, los *scripts* de *shell* se pueden invocar con el parámetro *-v*, que muestra cada línea antes de ejecutarla.
- ▶ **Probar en archivos locales.** Antes de ejecutar el *script* en un entorno real, conviene probar con archivos de prueba.
- ▶ **Empezar a probar con casos pequeños.** Por ejemplo, si el *script* va a operar sobre una lista de elementos, conviene probar que la tarea sobre un elemento funciona correctamente antes de introducir la lógica del bucle.
- ▶ **No olvidar probar las situaciones límite.** Siguiendo con el caso anterior, conviene probar que el *script* puede operar con un elemento, con muchos elementos y que no falla si la lista es de cero elementos.
- ▶ **Asumir que algo va a ir mal.** Cuanto más control de errores incorpore el *script*, más estable será. El control de errores implica tanto detectarlos como la lógica necesaria para tratar el error: un mensaje de error legible para el usuario, deshacer las acciones llevadas a cabo hasta el momento, etc.
- ▶ **Generalizar.** Un *script* más general es más fácil de extender a otros casos, probablemente será más fácil de probar y, por tanto, más robusto.

2.8. Acceso remoto en la nube

Los servidores en la nube no se instalan, **se despliegan**. Las instancias, como generalmente se denominan en muchos proveedores, se pueden arrancar de diversas maneras: en una consola web, con una herramienta de línea de comandos o a través de aplicaciones propias que usen un SDK del proveedor. Esta versatilidad ofrece posibilidades a los administradores para automatizar los despliegues al máximo. Los siguientes apartados muestran un despliegue manual en AWS EC2 con el objetivo de familiarizarse con los conceptos de este servicio.

EC2, o Elastic Compute Cloud, permite que los usuarios alquilen recursos de computación por segundos a partir de máquinas virtuales en varios sistemas operativos. Estas instancias se pueden personalizar para ejecutar cualquier aplicación que el usuario necesite.

La consola de administración de AWS es la cara pública de EC2 y de todos los servicios de AWS. Para desplegar una nueva instancia en la consola, hay que abrir la sección de EC2 en el menú Services y hacer *click* en Launch Instance. Se abrirá un asistente de despliegue en el que se podrá seleccionar el sistema operativo, tal como muestra la Figura 10.

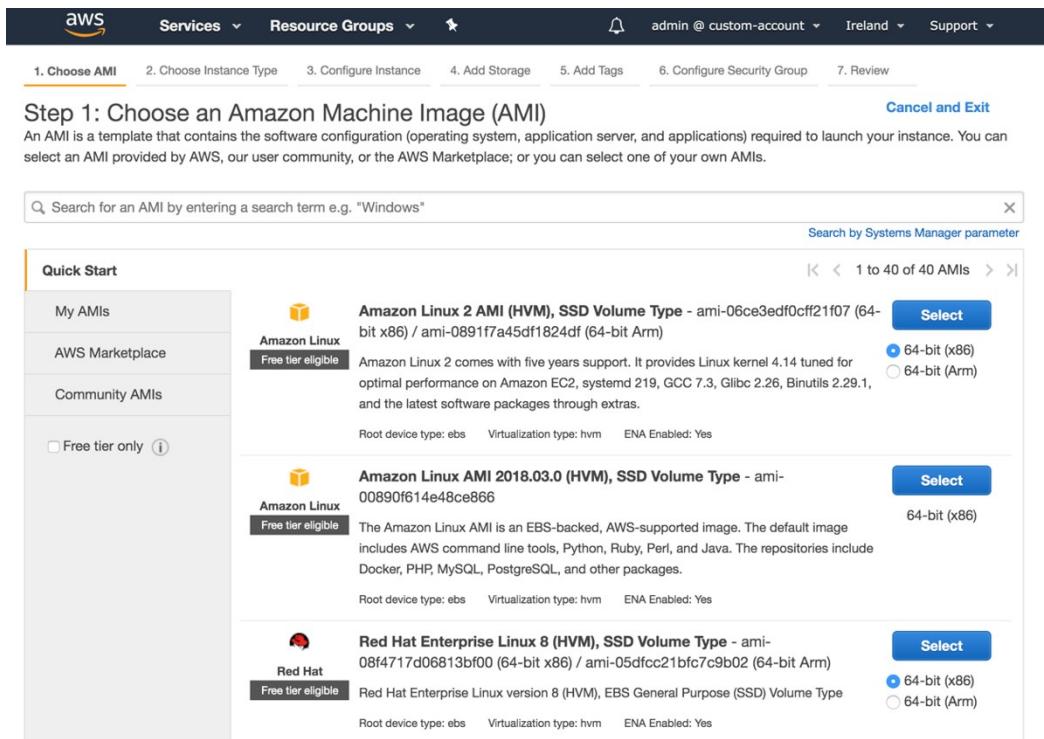


Figura 10. Consola de EC2, selección de sistema operativo. Fuente: elaboración propia.

Este paso no ofrece un sistema operativo como tal, sino una imagen de máquina o **AMI** (Amazon *machine image*). Las AMI se usan para arrancar máquinas, ya tienen los paquetes de *software* instalados, configurados y listos para ejecutarse. Amazon provee AMI para diversos sistemas operativos y la oferta crece en las pestañas del AWS Marketplace y Community AMIs.

Por ejemplo, Canonical proporciona imágenes soportadas oficialmente de varias versiones de Ubuntu, mientras que el AWS Marketplace ofrece *appliances* virtuales o máquina con *software* preinstalado para ejecutar, por ejemplo, un *router* virtual o un CMS corporativo. Algunas de estas AMI son gratuitas (es decir, sin coste por encima del precio de la instancia de AWS), mientras que otras tienen un sobrecoste. Este asistente también permite escoger una AMI propia.

Para demostrar la conexión remota a un equipo Linux, se escoge la imagen de **Amazon Linux 2**. El siguiente paso consiste en la elección del tipo de instancia. Las instancias EC2 pueden tener diversas configuraciones y especificaciones de *hardware* virtual. Además de varios tamaños de memoria y CPU (*central processing unit*), los

tipos de instancias, también denominados *flavors* informalmente, ofrecen diferentes configuraciones de velocidad de almacenamiento, rendimiento de tarjeta de red, GPU e incluso procesadores propios de AWS optimizados para, por ejemplo, la inferencia en aplicaciones de aprendizaje automático. Para este ejemplo, se selecciona el tamaño más pequeño, t2.nano, como en la Figura 11.

The screenshot shows the AWS Step 2: Choose an Instance Type wizard. The top navigation bar includes 'Services', 'Resource Groups', 'admin @ custom-account', 'Ireland', and 'Support'. Below the navigation, a progress bar shows steps 1. Choose AMI, 2. Choose Instance Type (which is highlighted in blue), 3. Configure Instance, 4. Add Storage, 5. Add Tags, 6. Configure Security Group, and 7. Review. A sub-progress bar below the main one shows steps 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance, 4. Add Storage, 5. Add Tags, 6. Configure Security Group, and 7. Review. A message 'Step 2: Choose an Instance Type' is displayed, followed by a note about Amazon EC2 providing a wide selection of instance types optimized for different use cases. A 'Learn more' link is provided. Below this, a table lists various instance types under 'General purpose' family, filtered by 'All instance types' and 'Current generation'. The table columns include Family, Type, vCPUs, Memory (GiB), Instance Storage (GB), EBS-Optimized Available, Network Performance, and IPv6 Support. The 't2.nano' row is selected, indicated by a blue border and checked checkbox in the first column. Other rows listed are t2.micro (Free tier eligible), t2.small, t2.medium, t2.large, t2.xlarge, and t2.2xlarge.

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	IPv6 Support
<input checked="" type="checkbox"/>	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.micro <small>Free tier eligible</small>	1	1	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.small	1	2	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.large	2	8	EBS only	-	Low to Moderate	Yes
<input type="checkbox"/>	General purpose	t2.xlarge	4	16	EBS only	-	Moderate	Yes
<input type="checkbox"/>	General purpose	t2.2xlarge	8	32	EBS only	-	Moderate	Yes

Figura 11. Tipo de instancia o *flavor*. Fuente: elaboración propia.

Una vez seleccionado el tamaño, el siguiente paso permite configurar detalles de la instancia. En la parte superior, se selecciona la red virtual y la subred a la que conectar la instancia. Además, y dado que el objetivo es demostrar el acceso remoto, la opción de asignar una IP pública está habilitada. Las redes virtuales suelen tener rangos de direcciones privadas, personalizables por el usuario, pero no enruteables desde Internet. Esto no suele ser un problema, y de hecho suele ser deseable. AWS ofrece varios servicios para exponer una aplicación a Internet (por ejemplo, API Gateway), pero, en este caso, es suficiente con que la instancia tenga una IP pública. La tarjeta de red de la instancia seguirá teniendo una IP privada y AWS configurará un NAT (*network address translation*) entre la IP pública y la privada.

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances: 1

Purchasing option: Request Spot instances

Network: vpc-08a3dd8391df4d996 | custom

Subnet: subnet-0b72d64a627e30068 | subnet1 | eu-west-1c 251 IP Addresses available

Auto-assign Public IP: Enable

Placement group: Add instance to placement group

Capacity Reservation: Open

IAM role: None

Figura 12. Selección de red. Fuente: elaboración propia.

En la parte inferior de este diálogo, aparece una sección importante: **User data** (ver Figura 13). Es una característica fundamental de EC2, ya que proporciona la base para la automatización de despliegues: el contenido del campo User data se ejecutará durante el arranque, por lo que se puede personalizar una AMI genérica para que funcione como, por ejemplo, un servidor web o un servidor de base de datos, simplemente arrancando una instancia. Las AMI de Ubuntu y Amazon Linux soportan *scripts* de *shell* en este campo, mientras que las AMI de Windows soportan PowerShell. En el ejemplo de la Figura 13, no se hace más que escribir un mensaje en un archivo, pero cualquier tarea que se haya automatizado en un *script* podría ejecutarse desde el User data.

Advanced Details

Metadata accessible: Enabled

Metadata version: V1 and V2 (token optional)

Metadata token response hop limit: 1

User data:

As text As file Input is already base64 encoded

```
mkdir -p /opt/app
echo "Hello World" > /opt/app/message
```

Figura 13. Apartado User data. Fuente: elaboración propia.

Los apartados de almacenamiento y etiquetas no aportan demasiado valor para el ejemplo de acceso remoto, así que no se van a comentar en detalle. El paso de los grupos de seguridad, sin embargo, sí es importante.

Los **grupos de seguridad**, o *security groups*, son *firewalls* a nivel de instancia que se ejecutan a nivel de hipervisor, por lo que liberan al sistema operativo de ejecutar un *firewall* interno. Los grupos de seguridad de AWS bloquean todo el tráfico entrante y permiten todo el tráfico saliente por defecto. Dado que el objetivo es el acceso remoto a un equipo Linux, es necesario añadir al menos una regla para permitir el SSH desde el equipo de origen. Se puede abrir el puerto a cualquier IP, aunque es recomendable restringirlo a la IP que detecta la consola web, por ejemplo, para limitar más el acceso, tal como muestra la Figura 14.

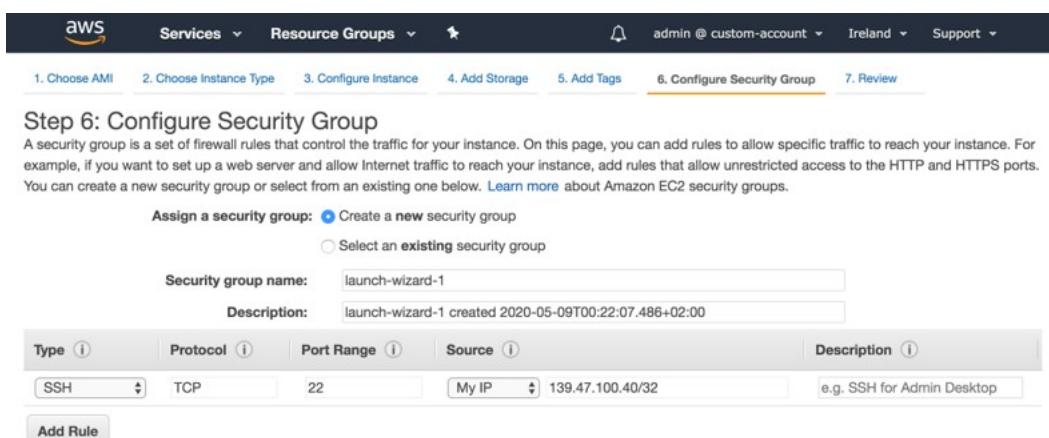


Figura 14. Grupos de seguridad. Fuente: elaboración propia.

El último paso es una mera revisión de las opciones seleccionadas hasta el momento. En el momento de hacer *click* en Launch. Sin embargo, el asistente ofrece la opción de **escoger un par de claves**, o *key pair*. Las claves permiten el acceso seguro a la instancia, tal como se ha visto. Las AMI oficiales no tienen contraseña por defecto y la única manera de conseguir acceso remoto es mediante un par de claves. La clave privada se descarga en el momento de la creación y AWS no la guarda, por lo que no es posible recuperarla si se pierde. AWS solo mantiene la parte pública y se encarga de copiarla a la instancia durante el arranque. Varias instancias pueden compartir un mismo par de claves.

En caso de crear una clave nueva al vuelo, es necesario descargar el archivo .pem y guardarlo en el disco, tal como describe la Figura 15. Una vez descargado, hay que configurar los permisos del archivo solo como lectura; por ejemplo, con chmod 400 fichero.pem, en MacOS o en Linux.

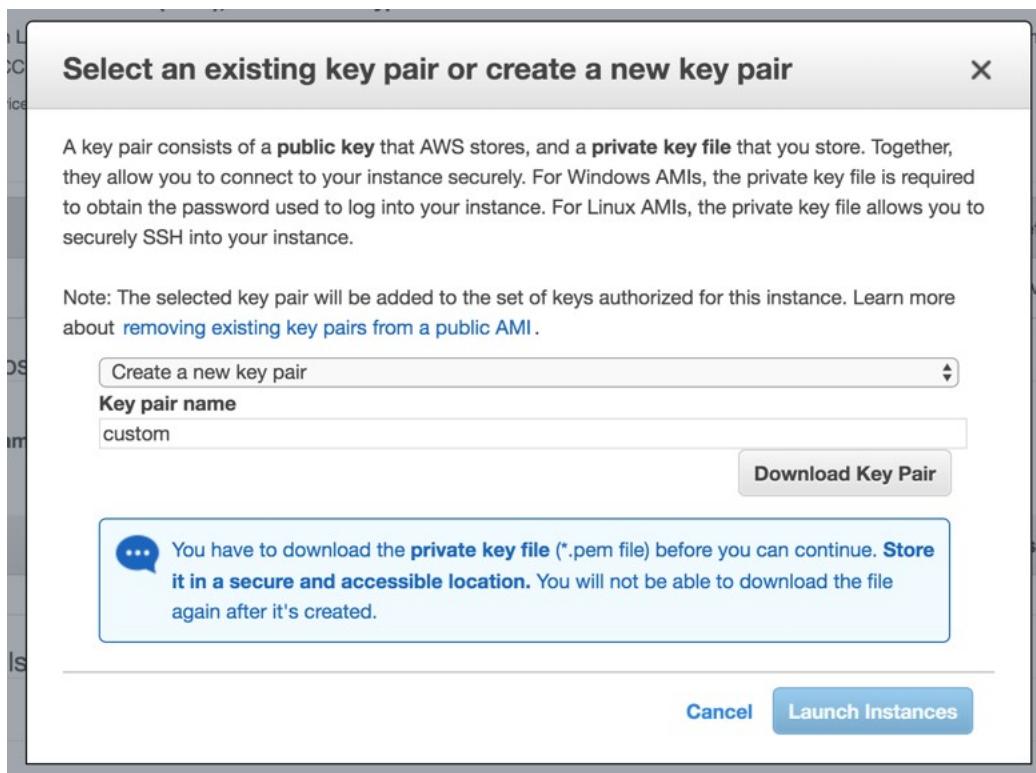


Figura 15. Creación de par de claves. Fuente: elaboración propia.

Una vez seleccionada la clave, o tras crear una nueva y descargar el archivo .pem, AWS inicia el despliegue de la instancia. En cuestión de segundos, cambiará a estado *running* y será posible obtener la IP pública. En la Figura 16, la IP pública es 63.35.201.236., mientras que la IP privada, que no es necesario conocer para este acceso, es 172.31.12.135.

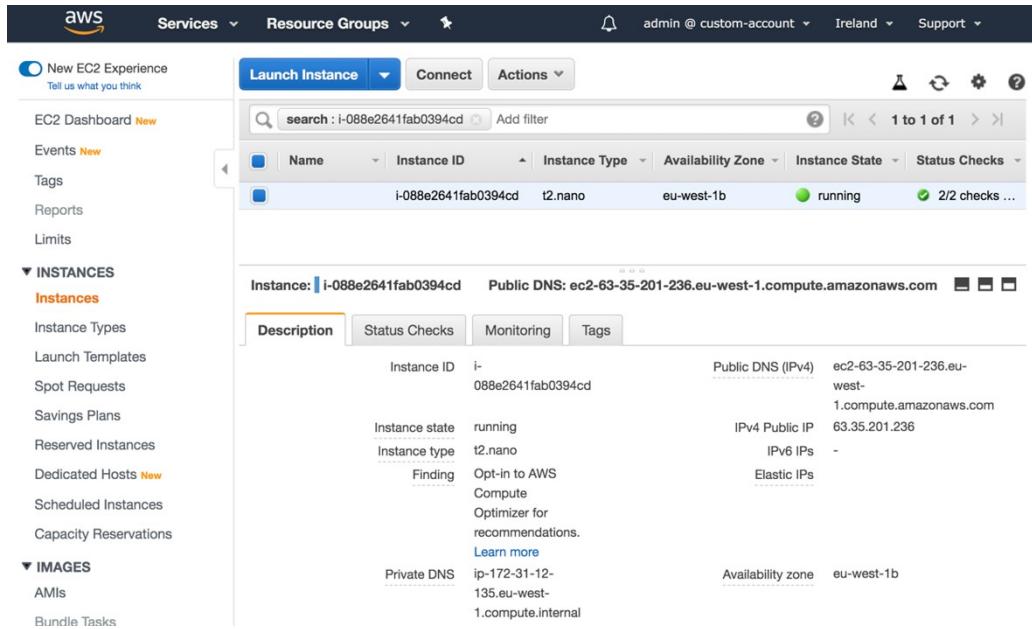


Figura 16. Instancia EC2 arrancada. Fuente: elaboración propia.

Con la instancia arrancada, ya es posible acceder por SSH con el usuario ec2-user. Este usuario existe por defecto en las AMI de Amazon Linux y está disponible en la documentación. Otras AMI tendrán otros nombres de usuario. El comando para iniciar sesión será ssh -i custom.pem ec2-user@63.35.201.236. Tal como muestra la Figura 17, será necesario aceptar la **firma de la clave**, al igual que al acceder a la instalación de Ubuntu de los apartados anteriores.

```
1. ec2-user@ip-172-31-12-135:~  
robermac:~ ac18101$ ssh -i custom.pem ec2-user@63.35.201.236  
The authenticity of host '63.35.201.236 (63.35.201.236)' can't be established.  
ECDSA key fingerprint is SHA256:W/wH9M6n4+IjzHhJqTFBeeTspKscQJSg6zNh89E5wfo.  
Are you sure you want to continue connecting (yes/no)? yes  
Warning: Permanently added '63.35.201.236' (ECDSA) to the list of known hosts.  
  
_I _I )  
_I ( / Amazon Linux 2 AMI  
  
https://aws.amazon.com/amazon-linux-2/  
No packages needed for security; 9 packages available  
Run "sudo yum update" to apply all updates.  
-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file or directory  
[ec2-user@ip-172-31-12-135 ~]$ |
```

Figura 17. Conexión SSH. Fuente: elaboración propia.

Las instancias de AWS ofrecen un servicio de metadatos integrado en una pequeña API (*application programming interface*) estática accesible por HTTP. Desde la propia instancia, se pueden recuperar datos, como el ID de la instancia o de la AMI, con una simple llamada de curl, como muestra la Figura 18.



```
1. ec2-user@ip-172-31-12-135:~  
[ec2-user@ip-172-31-12-135 ~]$ curl http://169.254.169.254/1.0/meta-data/instance-id  
i-088e2641fab0394cd[ec2-user@ip-172-31-12-135 ~]$  
[ec2-user@ip-172-31-12-135 ~]$ curl http://169.254.169.254/1.0/meta-data/ami-id  
ami-06ce3edf0cff21f07[ec2-user@ip-172-31-12-135 ~]$  
[ec2-user@ip-172-31-12-135 ~]$
```

Figura 18. Acceso a los metadatos de la instancia. Fuente: elaboración propia.

2.9. Referencias bibliográficas

AWS. (s. f.). *Amazon Linux 2*. <https://aws.amazon.com/es/amazon-linux-2/>

Debian. (s. f.). *Información sobre Debian «buster»*.

<https://www.debian.org/releases/stable/>

Dulaney, E. (2018). *Linux All-in-one for Dummies* (6.ª ed.; cap. I, pp. 7-57). John Wiley & Sons.

Frisch, A.E. (2002). *Essential System Administration* (3.ª ed.; cap. XV, pp. 885-942). O'Reilly.

Matotek, D., Turnbull, J. y Lieverdink, P. (2017). *Pro Linux System Administration: Learn to Build Systems for Your Business Using Free and Open Source Software* (2.ª ed.; cap. X, pp. 417-471). Apress.

Página de CentOS (<https://www.centos.org/>).

Putty, (s. f.). *Download PuTTY*. <https://www.putty.org/>

RedHat Customer Portal. (s. f.). *Product Documentation for Red Hat Enterprise Linux 8*. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/8/

Ubuntu Releases. (s. f.). *Ubuntu 18.04.5 LTS (Bionic Beaver)*.

<http://releases.ubuntu.com/18.04/>

Van Vugt, S. (2015). *Beginning the Linux Command Line* (2.ª ed.; cap. I, pp. 1-26). Apress.

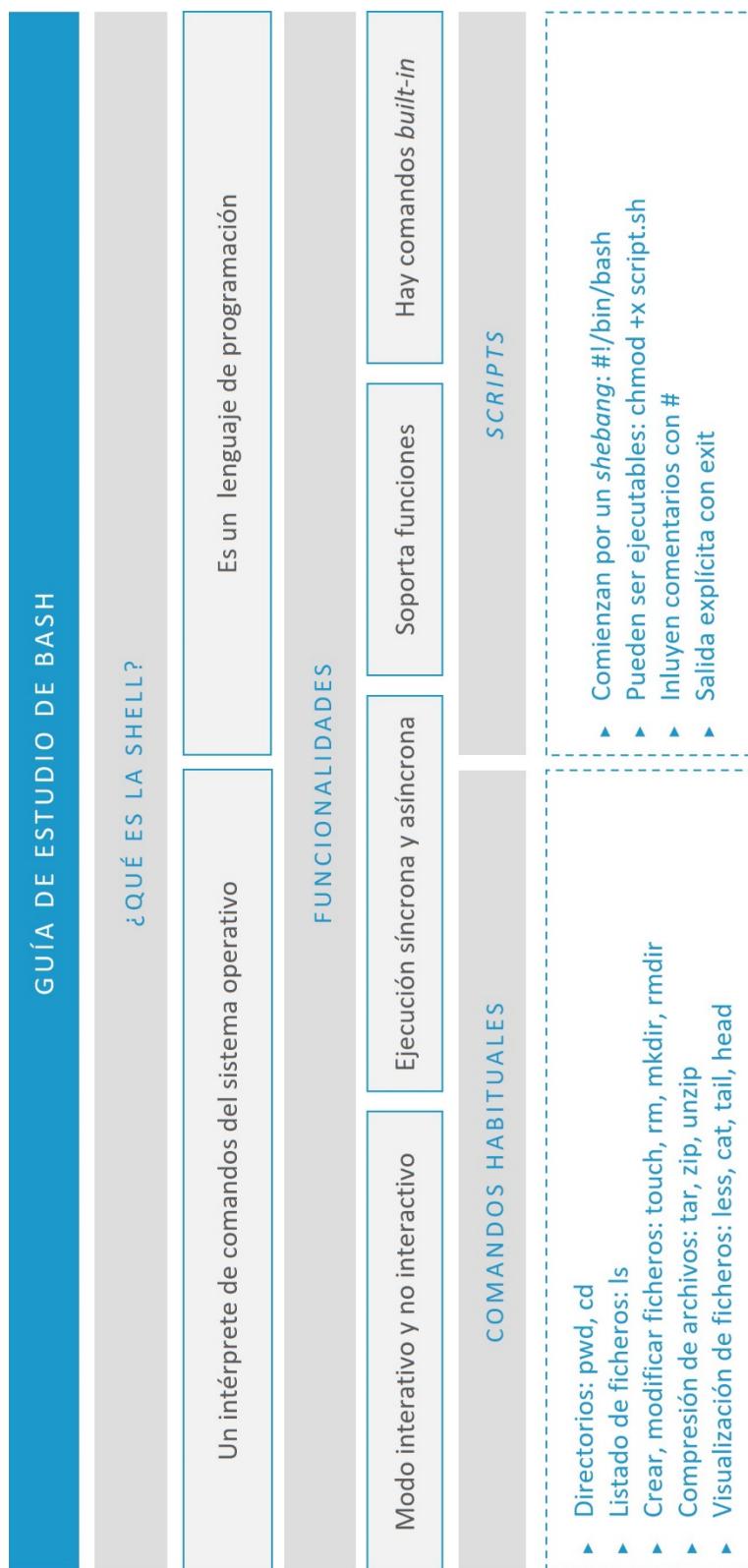
Administración de Sistemas en la Cloud

Guía de estudio de Bash

Índice

Esquema	3
Ideas clave	4
3.1. Introducción y objetivos	4
3.2. Qué es la <i>shell</i>	4
3.3. Comandos habituales	9
3.4. <i>Scripts</i> de Bash	24
3.5. Referencias bibliográficas	26

Esquema



3.1. Introducción y objetivos

Teniendo en cuenta el concepto de *script* y las ventajas que ofrece para la automatización de tareas de administración, este tema servirá de introducción para una de las herramientas de *scripting* más usadas en servidores Linux: la *shell* Bash.

Los **objetivos** que se pretenden conseguir son:

- ▶ Entender el concepto de *shell* y de *scripts* de *shell*.
- ▶ Aprender los comandos básicos de Bash.
- ▶ Familiarizarse con la ejecución de *scripts* de automatización.

3.2. Qué es la *shell*

Dicho de un modo muy sencillo, la *shell* es un **macroprocesador que ejecuta comandos**. Esta definición indica que hay una funcionalidad donde textos y símbolos se combinan para crear expresiones más grandes.

La *shell* es, a la vez, una intérprete de comandos y un lenguaje de programación. En su rol de intérprete de comandos, la *shell* ofrece al usuario una rica interfaz de utilidades o herramientas de GNU. Las características del lenguaje de programación, por su parte, permiten que estas herramientas se combinen.

Bash

Hay múltiples *shells* disponibles en entornos Linux y Unix. Aquí se usará Bash, ya que está disponible prácticamente la totalidad de las distribuciones. En caso de no estarlo, es relativamente sencillo traducir *scripts* de Bash a *scripts* de *shell* Bourne.

Historia de Bash

La *shell* Bourne original data de 1979 (Robbins, 2010), cuando se empezó a distribuir en entornos UNIX. Todavía se encuentra en /bin/sh en muchas distribuciones y, de hecho, ha evolucionado solo en contadas ocasiones. Otra *shell* de la época, la *shell* C (csh) ofrecía funcionalidades útiles para uso interactivo, como control de procesos e historial de comandos. Era habitual entre los administradores el uso de sh para programación de *scripts* y de csh para sesiones interactivas. Fue en 1983 cuando se presentó la *shell* Korn, o ksh, que se mantuvo compatible con sh, pero con funcionalidades interactivas de csh. Finalmente, Bash, de *bourne again shell*, apareció en 1989 como un clon de sh escrito desde cero, incorporando muchas funcionalidades de ksh desde entonces.

Características de la *shell*

La *shell* puede funcionar en modo interactivo y en modo no interactivo. En **modo interactivo**, la *shell* acepta entrada desde el teclado, ya sea el teclado local o un teclado remoto en una sesión SSH. El sistema operativo arranca una *shell* en el momento en el que un usuario inicia una nueva sesión (Van Vugt, 2015, pp. 1-26), ya sea una sesión local o por SSH. Por tanto, una *shell* no es sino un proceso más.

En **modo no interactivo**, la *shell* lee un archivo (es decir, un *script*) y ejecuta los comandos contenidos en él, línea por línea. Estos *scripts* pueden convertirse en comandos en sí mismos. Estos nuevos comandos tienen el mismo *status* que comandos del sistema en directorios como /bin, permitiendo que usuarios o grupos puedan construir sus propios entornos y automatizar sus tareas comunes.

En cuanto a los **comandos GNU**, la *shell* permite su ejecución de modo síncrono y asíncrono. En el primer caso, la *shell* acepta un comando, lo ejecuta y espera a que este termine antes de aceptar el siguiente comando. Por su parte, los comandos asíncronos continúan ejecutándose en paralelo con la *shell*, mientras lee y ejecuta comandos adicionales.

Las *shells* suelen incluir un pequeño **conjunto de comandos** que implementan funcionalidades que son casi imposibles de obtener por una vía diferente, por ejemplo, cd, break, continue y exec. Estos comandos no pueden ser implementados por fuera de la *shell*, porque la manipulan directamente. Otros comandos, como history, getopt, kill o pwd, pueden ser implementados por separado, pero, aun así, es más conveniente utilizarlos como comandos *built-in*. Algunos de estos comandos están orientados específicamente al uso interactivo, más que a aumentar el lenguaje de programación. Por ejemplo, la edición de línea de comando, el ya mencionado history, alias y los comandos de control de trabajo.

Funciones

Las funciones no son más que pequeñas subrutinas dentro de un *script* de *shell*. Son una forma de agrupar comandos para su ejecución posterior, usando un solo nombre para referenciar varias sentencias. Se ejecutan como un comando regular u ordinario. Las funciones de *shell* se ejecutan en el contexto de *shell* actual: no se crea ningún proceso nuevo para interpretarlos.

Parámetros

Un parámetro es una entidad que almacena valores. Puede ser un nombre, un número o un carácter especial. Una variable es un parámetro denotado por un nombre. Un parámetro se establece si se le ha asignado un valor.

Conceptos relacionados con la *shell*

- ▶ POSIX: su nombre es un acrónimo de *portable operating system interface* - Unix. Define una familia de estándares de sistemas abiertos basados en Unix, que intentan asegurar la portabilidad entre diferentes sistemas operativos.
- ▶ *Built-in*: es un comando que se implementa internamente por la *shell*, en lugar de por un programa ejecutable en algún lugar en el sistema de archivos. Es decir, es un comando llevado a cabo por la *shell*, como cd, en lugar de interpretarlo como una solicitud para cargar y ejecutar algún otro programa, como vim.

Esto tiene dos consecuencias principales. En primer lugar, por lo general, es más rápido, ya que el tiempo que toma cargar y ejecutar un programa es más prolongado. En segundo lugar, un comando *built-in* puede afectar el estado interno de la *shell*. Es por eso por lo que los comandos como cd deben ser *built-in*, dado que un programa externo no puede cambiar el directorio actual de la *shell*. Otros comandos, como echo, podrían ser *built-in* por un motivo de eficiencia, pero no hay ninguna otra razón intrínseca para que no puedan ser comandos externos.

- ▶ *Job control*: las características básicas del control de trabajos son la suspensión, continuación o terminación de procesos. El usuario puede ejecutarlo selectivamente, de acuerdo con sus necesidades.
- ▶ *Name*: también conocido como identificador, es una palabra que consiste únicamente de letras, números y guiones bajos y comienza con una letra o un guion bajo. Los nombres se utilizan para identificar variables y funciones.
- ▶ *Field*: una unidad de texto que es el resultado de una de las expansiones de la *shell*. Después de la expansión, al ejecutar un comando, los campos resultantes se utilizan como nombre del comando y argumentos.

- ▶ *Process group*: una colección o conjunto de procesos relacionados que tienen el mismo ID.
- ▶ *Process group ID*: identificador único que representa a un *process group* durante su ciclo de vida.
- ▶ *Token*: es una secuencia de caracteres que la *shell* considera una unidad. Puede ser una palabra o un operador.
- ▶ *Control operator*: es un *token* que tiene una función de control. Los más relevantes son:
 - (;): se pueden poner dos o más comandos en la misma línea separados por un punto y coma. Ambas series se ejecutarán secuencialmente y la *shell* esperará a que cada comando termine antes de iniciar el siguiente.
 - (&): cuando una línea termina con &, la *shell* no esperará a que termine el comando. El comando se ejecutará en segundo plano y el usuario podrá seguir introduciendo comandos.
 - (\$?): el código de salida de la orden anterior se almacena en el parámetro de shell \$. Este parámetro se utiliza para comprobar el estado de la última orden ejecutada. Si el valor devuelto por \$? es 0, significa que el último comando acabó con éxito; de lo contrario, el comando falló. El valor concreto depende de cada utilidad, pero, en todo caso, es un número entero de 8 bits.
 - (&&): un «*and*» lógico. El segundo comando se ejecuta solo si el primero tiene éxito. Por ejemplo, test "\$VAR1" = "val1" && echo \$VAR1 imprime el valor de la variable de entorno VAR1, solo si está definida como "val1".
 - (||): un «*or*» lógico. El segundo comando se ejecuta solo cuando la primera orden ha fallado. Por ejemplo, test "\$VAR1" != "val1" || echo "ERROR" imprime el mensaje de error si VAR1 no contiene "val1".
 - (#): todo lo escrito después de # es ignorado por la *shell*. Esto es útil para escribir un comentario y que no tenga efectos en la ejecución.

3.3. Comandos habituales

Este apartado muestra algunos de los ejemplos de expresiones habituales de *shell*, tanto de comandos básicos como con ciertos parámetros útiles.

A continuación, en el vídeo *Primeros pasos en Bash*, se verá una demostración de comandos básicos en Bash: archivos y directorios, utilidades GNU, Sudo.



Accede al vídeo

► Comandos de directorio:

- `pwd`: muestra el camino completo del directorio actual.

```
$ pwd
```

```
/etc/apache2/extra
```

- `cd`: cambia de directorio.

```
$ cd /etc/apache2
```

- `cd ~` : lleva a la carpeta *home* del usuario.

- `cd -` : lleva a la última ruta.

- `cd ..` : cambia al directorio padre del directorio actual.

► Listado de ficheros:

- `ls`: listado de ficheros.

```
$ ls
```

```
extra          magic          other
httpd.conf      mime.types      users
httpd.conf.pre-update    original
```

- `ls -al`: lista ficheros, carpetas e información.

```
$ ls -al
```

```
total 128
drwxr-xr-x  10 root  wheel   320 Aug 27  2018 .
drwxr-xr-x  90 root  wheel  2880 Apr 18 17:22 ..
drwxr-xr-x  14 root  wheel   448 Apr  4  2018 extra
-rw-r--r--   1 root  wheel 21150 Aug 27  2018 httpd.conf
```

```
-rw-r--r-- 1 root wheel 21150 Apr  4 2018 httpd.conf.pre-update
-rw-r--r-- 1 root wheel 13077 Apr  4 2018 magic
-rw-r--r-- 1 root wheel 61118 Apr  4 2018 mime.types
drwxr-xr-x 4 root wheel   128 Apr  4 2018 original
drwxr-xr-x 3 root wheel    96 Apr  4 2018 other
drwxr-xr-x 3 root wheel    96 Aug 27 2018 users
```

- `ls -aR`: lista de forma recursiva.
- `ls -aR | more`: lista de forma recursiva, usando una canalización para mostrarlo por pantalla. Es útil si la salida ocupa más líneas que las que puede mostrar la ventana de consola.
- `ls -alR > resultado.txt`: similar a la anterior, pero vuelca el resultado de la salida a un archivo, en vez de mostrarlo por pantalla.
- `ls *.htm`: enumera todos los archivos terminados en .htm.
- `ls -al dir/subdir/`: muestra un listado de los ficheros en ese directorio.

► Crear, modificar y borrar ficheros y carpetas:

- `touch /home/usr/html/index.htm`: crea el fichero index.htm sin contenido.
- `rm file.txt`: borra file.txt.
- `rm -rf dir/`: borra el directorio de forma recursiva y sin confirmación.
- `mkdir carpeta`: crea un directorio con el nombre carpeta.
- `rmdir carpeta`: borra el directorio llamado carpeta.

► Archivos comprimidos:

- `zip arch.zip /home/usr/public/dir`: comprime el directorio y su contenido en el fichero arch.zip.
- `unzip arch.zip`: descomprime arch.zip.
- `unzip -v arch.zip`: visualiza el contenido de arch.zip.
- `tar xvzf package.tgz`: descomprime el fichero package.tgz.

► Visualización de ficheros:

- `less fichero.log`: muestra el contenido del fichero solo en modo lectura; permite búsquedas, ir al principio y al final del fichero, pasar de página, etc.
- `cat fichero.log`: vuelca el contenido completo del fichero en la pantalla.

- `tail fichero.log`: muestra las últimas diez líneas del fichero.
- `tail -5 fichero.log`: muestra las últimas cinco líneas del fichero.
- `tail -f fichero.log`: muestra las últimas diez líneas del fichero y se engancha al fichero, mostrando las líneas nuevas que aparezcan en este. Es muy útil para mostrar un fichero de *log*, según se actualiza.
- `head fichero.log`: muestra las diez primeras líneas del fichero.
- `tail -5 fichero.log`: muestra las últimas cinco líneas del fichero.

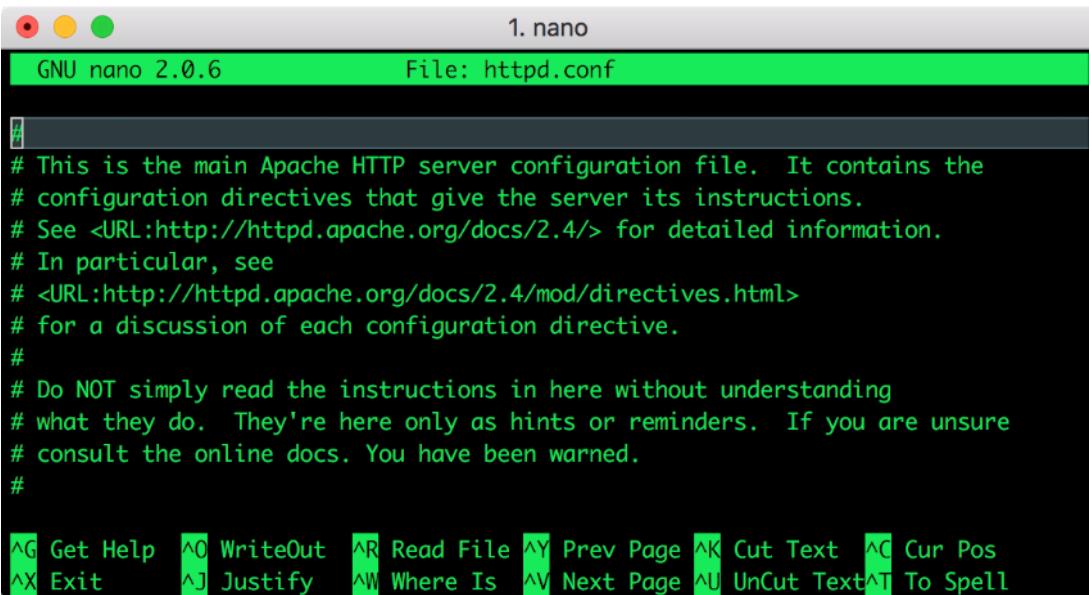
► Otros comandos:

- `cp -a /home/usr/origen/* /home/usr/destino/`: copia todos los contenidos de un directorio a otro, manteniendo sus permisos.
- `du -sh`: visualiza el espacio total ocupado por la carpeta actual.
- `du -sh *`: muestra el espacio ocupado de cada fichero.
- `whoami`: muestra el nombre del usuario.
- `cat /var/log/secure* | grep Accepted | awk '{print $9 " " $11}' | sort | uniq -c | sort -rn | head -20`: visualiza las veinte IP con más *login* exitosos.
- `cat /var/log/secure* | grep "Failed password" | awk '{print $9 " " $11}' | sort | uniq -c | sort -rn | head -20`: visualiza las veinte IP con más intentos de *login* incorrectos.

Editores

Aunque los editores enfocados a programación ofrecen multitud de características para facilitar la vida del administrador o del desarrollador, es habitual encontrarse en la situación de tener que **editar o crear un fichero** en un servidor sin poder usar una herramienta de entorno gráfico. Prácticamente cualquier distribución incorporará al menos un editor en modo texto, es decir, un editor de ficheros que funciona en la consola de línea de comandos. Entre estos editores, se encuentran `vi`, `vim`, `nano`, `pico` o `emacs`, por mencionar algunos.

Al no disponer de entorno gráfico, estos editores se manejan con **combinaciones de teclas**. Por ejemplo, nano muestra una barra en la parte inferior con los comandos más habituales. En la Figura 1, $\wedge O$ significa Ctrl+O.



The screenshot shows a terminal window titled "1. nano" with the command "GNU nano 2.0.6" and the file path "File: httpd.conf". The main area displays the Apache configuration file content:

```
# This is the main Apache HTTP server configuration file. It contains the
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs/2.4/> for detailed information.
# In particular, see
# <URL:http://httpd.apache.org/docs/2.4/mod/directives.html>
# for a discussion of each configuration directive.
#
# Do NOT simply read the instructions in here without understanding
# what they do. They're here only as hints or reminders. If you are unsure
# consult the online docs. You have been warned.
#
```

At the bottom, there is a menu of keyboard shortcuts:

$\wedge G$	Get Help	$\wedge O$	WriteOut	$\wedge R$	Read File	$\wedge Y$	Prev Page	$\wedge K$	Cut Text	$\wedge C$	Cur Pos
$\wedge X$	Exit	$\wedge J$	Justify	$\wedge W$	Where Is	$\wedge V$	Next Page	$\wedge U$	UnCut Text	$\wedge T$	To Spell

Figura 1. Editor nano con el fichero httpd.conf. Fuente: elaboración propia.

Uno de los editores más extendidos es vi (Van Vugt, 2015, pp. 69-90). Este editor funciona con el concepto de «modo»: por defecto, arranca en modo de «comando», en el que se pueden ejecutar operaciones. Para poder escribir, hay que pulsar *i* para entrar en modo de «edición». Habrá que pulsar *Esc* para volver al modo de comando para, por ejemplo, guardar el fichero o salir de vi.

```
#  
# This is the main Apache HTTP server configuration file. It contains the  
# configuration directives that give the server its instructions.  
# See <URL:http://httpd.apache.org/docs/2.4/> for detailed information.  
# In particular, see  
# <URL:http://httpd.apache.org/docs/2.4/mod/directives.html>  
# for a discussion of each configuration directive.  
#  
# Do NOT simply read the instructions in here without understanding  
# what they do. They're here only as hints or reminders. If you are unsure  
# consult the online docs. You have been warned.  
#  
# Configuration and logfile names: If the filenames you specify for many  
# of the server's control files begin with "/" (or "drive:/" for Win32), the  
# server will use that explicit path. If the filenames do *not* begin  
# with "/", the value of ServerRoot is prepended -- so "logs/access_log"  
:wq
```

Figura 2. Editor vi con el fichero httpd.conf en modo comando con el comando wq. Fuente: elaboración propia.

Redirección

Bash soporta la redirección de la salida de un comando. Por un lado, puede volcar la salida estándar en un fichero:

```
$ ls -la > listado.txt
```

Este comando no muestra nada por pantalla. El contenido se escribe en el fichero listado.txt, sobrescribiendo cualquier contenido anterior. También es posible **añadir** contenido al final del contenido actual del fichero:

```
$ ls -la >> listado.txt
```

Por otro lado, se puede usar el resultado de un comando como entrada de otro usando la funcionalidad de **tubería** o *pipe*. Por ejemplo, el siguiente comando muestra el contenido del fichero de configuración del servidor SSH, pero muestra solo las líneas no comentadas y ordenadas por orden alfabético:

```
$ cat sshd_config | grep -v "#" | sort | uniq
```

La redirección en sentido contrario también está soportada, es decir, se puede leer el contenido de un fichero y pasarlo por la entrada estándar a un comando. Por ejemplo, la siguiente línea lee el fichero `input.txt` y lo entrega por la entrada estándar a `cat`. El resultado es idéntico a ejecutar `cat input.txt` directamente, pero la redirección de entrada se puede usar con comandos que no estén preparados para leer ficheros directamente.

```
$ cat < listado.txt
```

Sudo

No todos los usuarios tienen los mismos privilegios en Linux: hay un usuario administrador inicial, `root`, capaz de realizar cualquier tarea. Otros usuarios solo pueden acceder a sus ficheros y ejecutar comandos no administrativos, a menos que reciban privilegios adicionales, bien directamente o a través de un grupo.

Para ejecutar operaciones administrativas, no hay más que iniciar sesión como `root`. Sin embargo, esto conlleva ciertos riesgos de seguridad, como que el usuario teclee un comando sintácticamente válido, pero que no hace lo que debe, borrando datos importantes por error (Van Vugt, 2015, pp. 179-196). Es habitual que la cuenta de `root` esté deshabilitada por defecto y que ni siquiera tenga contraseña, por lo que no se puede iniciar sesión con ella en muchas distribuciones.

La alternativa es usar el mecanismo de Sudo. La idea es que tareas de administración concretas pueden asignarse a usuarios específicos. Si un usuario necesita ejecutar una tarea administrativa como, por ejemplo, instalar un paquete nuevo con `apt-get install`, debe ejecutarlo con Sudo:

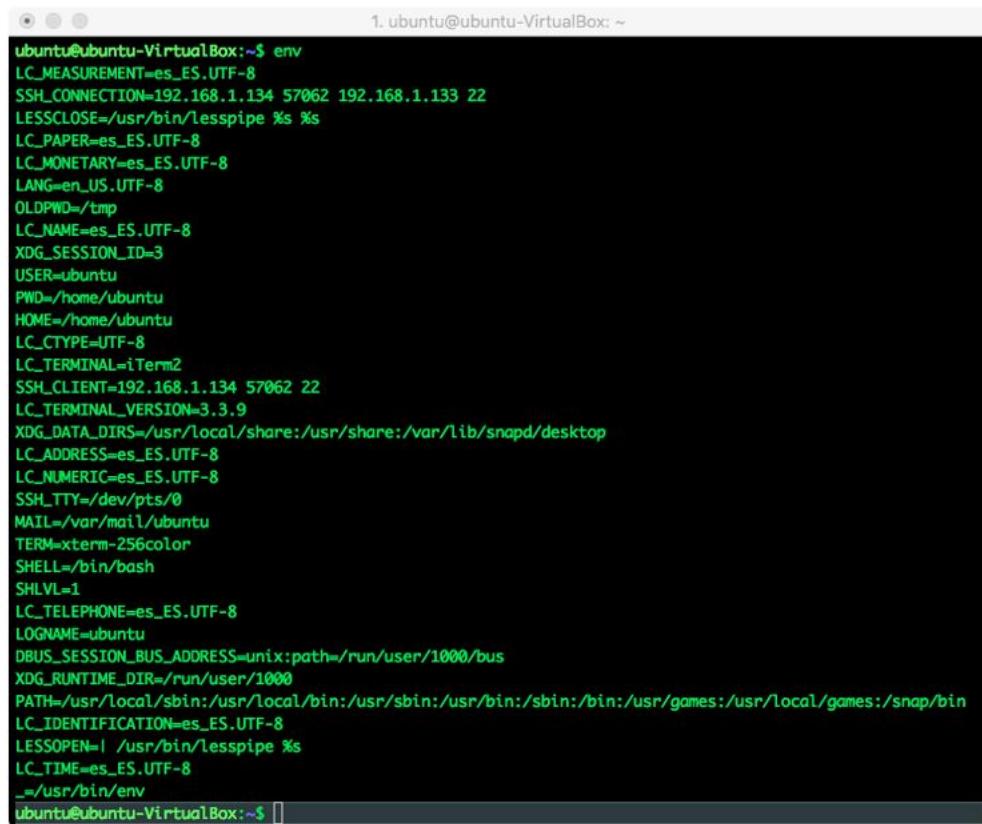
```
sudo apt-get install openssh-server
```

El usuario `root` podría ejecutar el mismo comando sin Sudo, pero es obligatorio para cualquier otro usuario. Sudo solicita la contraseña del usuario y rechaza el comando

si el usuario no tiene permiso para ejecutarlo. El fichero sudoers contiene los permisos asignados a los usuarios y grupos del sistema. Debe editarse con visudo: este comando abre el fichero sudoers con el editor del sistema (por ejemplo, vi o nano) y lo valida al guardarlo. De esta manera, se evita que un usuario cometa un error de sintaxis al guardarla, rompiendo la funcionalidad de Sudo.

Variables de entorno

Cualquier proceso en Linux tiene asociado un conjunto de variables de entorno o *environment variables* (Dulaney, 2018). Las variables de entorno no son más que un nombre asociado a una **cadena de texto**. El comando env muestra las variables definidas en el entorno de la *shell*:

A screenshot of a terminal window titled "1. ubuntu@ubuntu-VirtualBox: ~". The window displays the output of the "env" command, which lists numerous environment variables. The variables include LC_MEASUREMENT, SSH_CONNECTION, LESSCLOSE, LC_PAPER, LC_MONETARY, LANG, OLDPWD, LC_NAME, XDG_SESSION_ID, USER, PWD, HOME, LC_CTYPE, LC_TERMINAL, SSH_CLIENT, LC_TERMINAL_VERSION, XDG_DATA_DIRS, LC_ADDRESS, LC_NUMERIC, SSH_TTY, MAIL, TERM, SHELL, SHLVL, LC_TELEPHONE, LOGNAME, DBUS_SESSION_BUS_ADDRESS, XDG_RUNTIME_DIR, PATH, LC_IDENTIFICATION, LESSOPEN, LC_TIME, and _=/usr/bin/env. The text is in white on a black background.

```
ubuntu@ubuntu-VirtualBox:~$ env
LC_MEASUREMENT=es_ES.UTF-8
SSH_CONNECTION=192.168.1.134 57062 192.168.1.133 22
LESSCLOSE=/usr/bin/lesspipe %s %s
LC_PAPER=es_ES.UTF-8
LC_MONETARY=es_ES.UTF-8
LANG=en_US.UTF-8
OLDPWD=/tmp
LC_NAME=es_ES.UTF-8
XDG_SESSION_ID=3
USER=ubuntu
PWD=/home/ubuntu
HOME=/home/ubuntu
LC_CTYPE=UTF-8
LC_TERMINAL=iTerm2
SSH_CLIENT=192.168.1.134 57062 22
LC_TERMINAL_VERSION=3.3.9
XDG_DATA_DIRS=/usr/local/share:/usr/share:/var/lib/snapd/desktop
LC_ADDRESS=es_ES.UTF-8
LC_NUMERIC=es_ES.UTF-8
SSH_TTY=/dev/pts/0
MAIL=/var/mail/ubuntu
TERM=xterm-256color
SHELL=/bin/bash
SHLVL=1
LC_TELEPHONE=es_ES.UTF-8
LOGNAME=ubuntu
DBUS_SESSION_BUS_ADDRESS=unix:path=/run/user/1000/bus
XDG_RUNTIME_DIR=/run/user/1000
PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/bin
LC_IDENTIFICATION=es_ES.UTF-8
LESSOPEN=-l /usr/bin/lesspipe %s
LC_TIME=es_ES.UTF-8
_=~/usr/bin/env
ubuntu@ubuntu-VirtualBox:~$ []
```

Figura 3. Variables definidas en el entorno de la *shell*. Fuente: elaboración propia.

Estas variables afectan al comportamiento de diversos elementos. Por ejemplo, la variable PATH define las rutas en las que la *shell* buscará un ejecutable cuando se

invoca un comando que no es una expresión interna de la *shell*, mientras que editores como vim usan la variable TERM para decidir cómo mostrar los archivos en la consola. Se puede consultar el valor de estas variables con echo:

```
echo $PATH
```

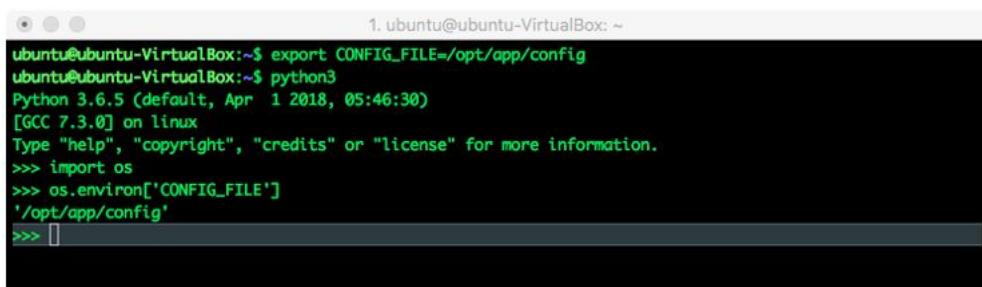
Se pueden crear nuevas variables, o cambiar valores de variables existentes, con export:

```
export CONFIG_FILE=/opt/app/config
```

Además de mostrarlas por pantalla, es posible usar el valor de una variable de entorno en un comando de Bash. La *shell* se encargará de recuperar el valor y sustituirlo.

```
$ mkdir /tmp/app
$ export CONFIG_FILE=/tmp/app/config
$ echo "DEBUG=true" > $CONFIG_FILE
$ cat $CONFIG_FILE
DEBUG=true
```

Al igual que la *shell* y los editores de línea de comandos, cualquier proceso puede acceder a sus variables de entorno. Por ejemplo, desde un proceso Python, el módulo os permite acceder a todas las variables (ver Figura 4).



The screenshot shows a terminal window titled 'ubuntu@ubuntu-VirtualBox: ~'. The user runs the command 'export CONFIG_FILE=/opt/app/config' followed by 'python3'. Inside the Python interpreter, the user imports the 'os' module and prints the value of 'os.environ['CONFIG_FILE']', which outputs '/opt/app/config'.

```
ubuntu@ubuntu-VirtualBox:~$ export CONFIG_FILE=/opt/app/config
ubuntu@ubuntu-VirtualBox:~$ python3
Python 3.6.5 (default, Apr 1 2018, 05:46:30)
[GCC 7.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.environ['CONFIG_FILE']
'/opt/app/config'
>>> []
```

Figura 4. Comandos Python para acceder a variables de entorno. Fuente: elaboración propia.

Las variables de entorno están muy extendidas como mecanismo de configuración, por ejemplo, en el caso de aplicaciones desplegadas como contenedores o en Kubernetes.

Heredocs

Los documentos *here*, o *heredocs*, son bloques de código que usan la redirección para alimentar un conjunto de líneas o comandos a un programa externo. Es habitual encontrarlos en *scripts* como método de insertar ficheros externos o de configuración en el propio fichero del *script*. De esta forma, es posible contener todo un *script* de automatización, incluyendo ficheros de configuración, en un único fichero.

El documento está limitado por dos cadenas de texto idénticas, la primera de ellas precedida de <<.

```
$ cat <<HEREDOC
Mensaje de varias lineas
que no podria imprimirse por pantalla
simplemente con "cat mensaje"
HEREDOC
```

Los documentos *here* se pueden combinar con tuberías y redirección.

```
$ cat > output.txt <<HEREDOC
Mensaje de varias lineas
que se guarda en un fichero
HEREDOC
```

La substitución de variables de entorno también está soportada. Además, para hacer más legibles los *scripts*, es posible tabular el contenido del *heredoc*, pero redirigirlo a la entrada del comando sin los caracteres de tabulación, con la expresión <<-.

```
$ NAME=World
$ cat > mensaje.txt <<-HELLOALL
```

```
Hello $NAME
Heredocs rule
HELLOALL
$ cat mensaje.txt
Hello World
Heredocs rule
```

Expresiones lógicas

El **comando test** evalúa una condición lógica y devuelve un código de salida cero, si la condición es verdadera, o distinto de cero, si es falsa. Este comando tiene una forma alternativa, en la que el nombre test se sustituye por corchetes, [], y otra forma que usa corchetes dobles, [[]]; esta segunda forma no aplica expansión de ruta ni división de palabras. Ambas formas se usan habitualmente en *scripts*, ya que permiten expresar condiciones con una sintaxis similar a otros lenguajes de programación.

```
$ test 1 -eq 2; echo $?
1
$ [[ 1 -eq 1 ]]; echo $?
0
```

Este comando se puede usar en conjunción con la expresión **if** de Bash, para construir bloques condicionales como en cualquier otro lenguaje. Admite el uso de **else** y debe terminarse con **fi**.

```
$ if [[ 1 -eq 1 ]] ; then
    echo igual
else
    echo diferente
fi
```

El comando **test** soporta condiciones lógicas para comprobar ficheros, variables de entorno, comparaciones de enteros, etc.

Expresiones de test	
	Significado
-e fichero	El fichero existe
-d directorio	El directorio existe y es un directorio
-f fichero	El fichero existe y es un archivo normal
-h enlace	El enlace existe y es un archivo de tipo enlace simbólico
-N fichero	El fichero existe y se modificó tras la última lectura
-o fichero	El fichero existe y el propietario es el usuario actual
-p tuberia	La tubería existe y es un archivo de tipo tubería
-r fichero	El fichero existe y se puede leer
-s fichero	El fichero existe y tiene un tamaño mayor que cero
-S socket	El socket existe y es un archivo de tipo socket
-w fichero	El fichero existe y se puede escribir
-x fichero	El fichero existe y es ejecutable
f1 -ef f2	Los ficheros f1 y f2 enlazan al mismo fichero
f1 -nt f2	El fichero f1 es más nuevo que f2
f1 -ot f2	El fichero f1 es más antiguo que f2
texto	El texto no es nulo
-n texto	El texto tiene una longitud mayor que cero
-z texto	El texto tiene una longitud cero
t1 == t2	Las cadenas de texto son idénticas. Usado dentro de [[]], t2 puede contener wildcards
t1 != t2	Las cadenas de texto no son idénticas. Usado dentro de [[]], t2 puede contener wildcards
t1 =~ reg	Las cadenas de texto t1 cumplen con la expresión regular reg. Solo se puede usar con [[]]
t1 < t2	La cadena t1 precede a t2
n1 -eq n2	Los enteros son iguales

Expresiones de test	
	Significado
n1 -ge n2	n1 es mayor o igual que n2
n1 -gt n2	n1 es mayor que n2
n1 -le n2	n1 es menor o igual que n2
n1 -lt n2	n1 es menor que n2
n1 -ne n2	Los enteros son diferentes
! condicion	Negación de la condicion
c1 && c2	«And» lógico con cortocircuito. Solo se puede usar en [[]]
c1 c2	«Or» lógico con cortocircuito. Solo se puede usar en [[]]

Tabla 1. Expresiones lógicas de test. Fuente: elaboración propia.

Las condiciones con cadenas de texto pueden dar lugar a problemas. Por ejemplo, si se quiere comprobar si una variable tiene un valor dado, este ejemplo podría fallar si \$VAR no está definido:

```
$ [ $VAR1 == yes ]; echo $?
-bash: [: ==: unary operator expected
2
```

Esto ocurre porque la sustitución de \$VAR por su valor convierte la expresión en [== yes], es decir, una expresión binaria sin el primer operador. En estos casos, es habitual rodear ambas cadenas con comillas e incluso añadir un carácter adicional en las dos cadenas, para evitar que una de las dos se convierta en una cadena vacía.

```
$ [ "$VAR"x == "yes"x ]; echo $?
1
$ VAR=yes
$ [ "$VAR"x == "yes"x ]; echo $?
0
```

Expansión y metacaracteres

Bash acepta metacaracteres para completar nombre de ficheros y directorios. Por ejemplo, `ls -l file` mostrará por consola los detalles del fichero `file`, si existe, pero `ls -l file*` mostrará los detalles de todos los ficheros que empiecen con `file`.

```
$ ls -l file*
-rw-r--r-- 1 ubuntu  ubuntu  0 May  9 10:11 file
-rw-r--r-- 1 ubuntu  ubuntu  0 May  9 10:12 file.log
-rw-r--r-- 1 ubuntu  ubuntu  0 May  9 10:12 file1
```

Metacaracteres			
	Significado	Ejemplo	Ficheros que aceptan el ejemplo
*	Cadena de cero o más caracteres	<code>ls -l file*</code>	<code>file, file1, filenew.log</code>
?	Cadena de exactamente un carácter	<code>ls -l file?.log</code>	<code>file0.log, fileX.log</code>
[abc]	Uno, cualquiera, de los caracteres indicados. El guion indica un rango (es decir, <code>a-z</code> aceptará cualquier letra en minúscula y <code>0-9</code> cualquier dígito)		
[a-z]		<code>file[0-9].log</code>	<code>file0.log, file1.log</code>
[0-9]			
[!abc]	Ninguno de los caracteres indicados	<code>file[0-9].log</code>	<code>fileX.loog</code>
~	Directorio <i>home</i> del usuario actual	<code>ls ~/Desktop</code>	
~usuario	Directorio <i>home</i> del usuario	<code>ls ~ubuntu/Desktop</code>	
~+	Directorio de trabajo actual	<code>ls ~+</code>	
~-	Directorio de trabajo anterior	<code>cp file +-</code>	

Tabla 2. Metacaracteres. Fuente: elaboración propia.

Además de los metacaracteres, Bash soporta la expansión de cadenas de texto, se conoce como *brace expansion*. Al contrario de los metacaracteres, que se usan

exclusivamente para nombres de archivo, la **expansión está orientada a texto**. Las palabras generadas en una expansión no tienen por qué coincidir con ficheros.

```
$ echo Rule{A1,A2,A3,B1,C1,C2}  
RuleA1 RuleA2 RuleA3 RuleB1 RuleC1 RuleC2
```

Además de la expansión del ejemplo anterior, Bash también soporta la expansión de rangos de números enteros.

```
$ echo Rule{1..5}  
Rule1 Rule2 Rule3 Rule4 Rule5
```

Bucles

La expansión de Bash se puede usar, por ejemplo, en un bucle.

```
$ for r in i{1..3};  
do  
    echo interacion $r  
done  
  
interacion i1  
interacion i2  
interacion i3
```

La lista de elementos puede ser cualquier *array*, por ejemplo, la lista de parámetros de la llamada al *script*, *\$@*.

```
$ cat file.sh  
for p in $@; do echo $p; done  
$ bash file.sh p1 param2 true  
p1  
param2  
true
```

También es posible generar el *script* al vuelo a partir de otro comando.

```
$ ls -l file?  
-rw-r--r-- 1 ubuntu  ubuntu  0 May  9 11:07 file1  
-rw-r--r-- 1 ubuntu  ubuntu  0 May  9 11:07 file2  
-rw-r--r-- 1 ubuntu  ubuntu  0 May  9 11:07 filex  
$ for f in $(ls file?); do echo Fichero $f; done  
Fichero file1  
Fichero file2  
Fichero filex
```

Finalmente, también se puede iterar sobre un rango de enteros, tal como en otros lenguajes.

```
$ for ((i=1; i <= 5; i += 1))  
do  
    if [[ -e file$i ]]; then  
        echo file$i existe;  
    else  
        echo file$i no existe  
    fi  
done  
  
file1 existe  
file2 existe  
file3 no existe  
file4 no existe  
file5 no existe
```

3.4. Scripts de Bash

Un *script* de Bash es un fichero de texto con una secuencia de comandos de *shell* (Van Vugt, 2015, pp. 319-351). No tiene más requisitos, así que un fichero con este contenido sería un *script* válido:

```
echo "Hello World"  
pwd
```

Este *script* se podría ejecutar invocándolo de la siguiente manera:

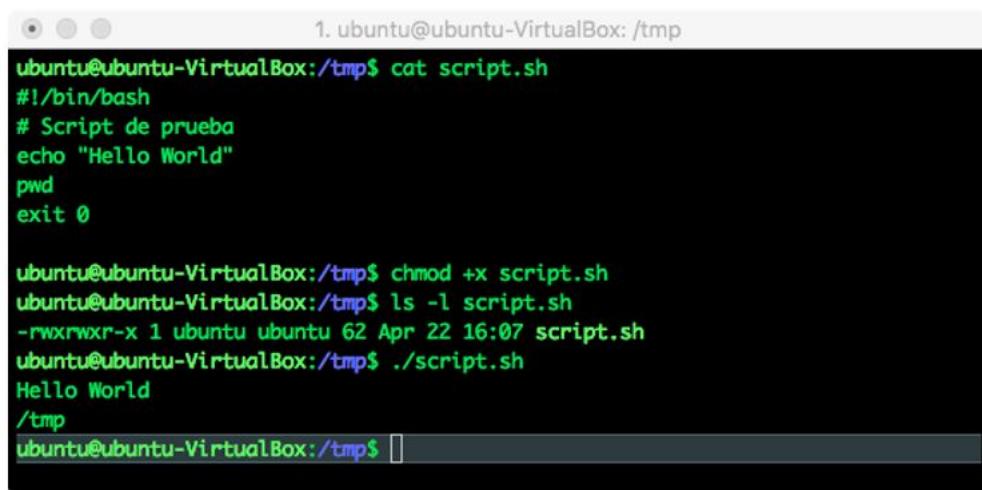
```
$ bash script.sh  
Hello World  
/etc/apache2
```

Sin embargo, conviene construir los *scripts* con las siguientes características:

- ▶ Deben empezar con un *shebang*, `#!`, en la primera línea, indicando la ruta al ejecutable de la *shell*. Esta línea es un comentario especial que permite ejecutar un *script* como un ejecutable binario más. Para un *script* de Bash, la línea sería `#!/bin/bash`, mientras que un *script* de Python podría usar `#!/usr/bin/python3` - v.
- ▶ Los *scripts* pueden incluir comentarios para facilitar la lectura del código por otros administradores.
- ▶ El uso de `exit` indica explícitamente cuando termina el *script*, tanto de manera correcta con un código de salida 0 como si hay algún error.
- ▶ Los *scripts* pueden ser considerados como ejecutables, de manera que se pueden invocar en la línea de comandos como si fueran un archivo compilado más. Esto se consigue habilitando el *flag* ejecutable de archivo con, por ejemplo, `chmod +x script.sh`.

Estas ideas se pueden aplicar al *script* anterior para obtener el siguiente:

```
#!/bin/bash
# Script de prueba
echo "Hello World"
pwd
exit 0
```



The screenshot shows a terminal window titled '1. ubuntu@ubuntu-VirtualBox: /tmp'. The user has run the command 'cat script.sh' to view the contents of the script. Then, they used 'chmod +x script.sh' to make it executable. After that, they ran 'ls -l script.sh' to show its permissions, which are set to '-rwxrwxr-x'. Finally, they executed the script with './script.sh', which printed 'Hello World' to the console.

```
ubuntu@ubuntu-VirtualBox:/tmp$ cat script.sh
#!/bin/bash
# Script de prueba
echo "Hello World"
pwd
exit 0

ubuntu@ubuntu-VirtualBox:/tmp$ chmod +x script.sh
ubuntu@ubuntu-VirtualBox:/tmp$ ls -l script.sh
-rwxrwxr-x 1 ubuntu ubuntu 62 Apr 22 16:07 script.sh
ubuntu@ubuntu-VirtualBox:/tmp$ ./script.sh
Hello World
/tmp
ubuntu@ubuntu-VirtualBox:/tmp$
```

Figura 5. Ejemplo de *script* en Bash. Fuente: elaboración propia.

Hasta ahora, se ha mostrado cómo ejecutar un *script* de dos maneras: como argumento del comando *bash* y convirtiendo el *script* en ejecutable e invocándolo directamente. Ambas opciones funcionan de manera similar: la *shell* desde la que se invoca arranca otra *shell* como un nuevo proceso y es esta la que se encarga de ejecutar los comandos del *script*.

Hay una tercera opción, ligeramente diferente: ejecutar el *script* en el mismo proceso de la *shell* actual. Este caso puede ser útil si el *script* cambia valores de variables de entorno que son necesarias en la *shell* actual (por ejemplo, para configurar la *shell* de alguna manera concreta). No obstante, puede tener efectos contraproducentes: si el *script* termina proactivamente con un comando *exit*, la *shell* actual también terminará. El comando para ejecutar los *scripts* en la propia *shell* es *source* o . (un punto):

```
$ source script.sh  
$ . script.sh
```

Scripts especiales

La *shell* ejecuta ciertos *scripts* durante el arranque. Concretamente, intentará leer los ficheros en este orden:

- `/etc/profile`. Se ejecuta durante el inicio de sesión.
- `~/.bash_profile`, o bien `~/.bash_login`, o bien `~/.profile`, el primero que encuentre. Se ejecuta durante el inicio de sesión.
- `~/.bashrc`. Se ejecuta en una *shell* sin inicio de sesión.

Estos *scripts* suelen contener opciones de configuración de las sesiones de usuario: definición de alias, el texto del *prompt* (el texto que se muestra en cada nueva línea antes de los comandos que introduce el usuario) y cualquier otra personalización que cada usuario quiera tener disponible.

3.5. Referencias bibliográficas

Dulaney, E. (2018). *Linux All-in-one for Dummies* (6.ª ed., cap. II, pp. 203-259). John Wiley & Sons.

Robbins, A. (2010). *Bash Pocket Reference* (apartado «History», pp. 2-3). O'Reilly Media.

Van Vugt, S. (2015). *Beginning the Linux Command Line* (2.ª ed.). Apress.

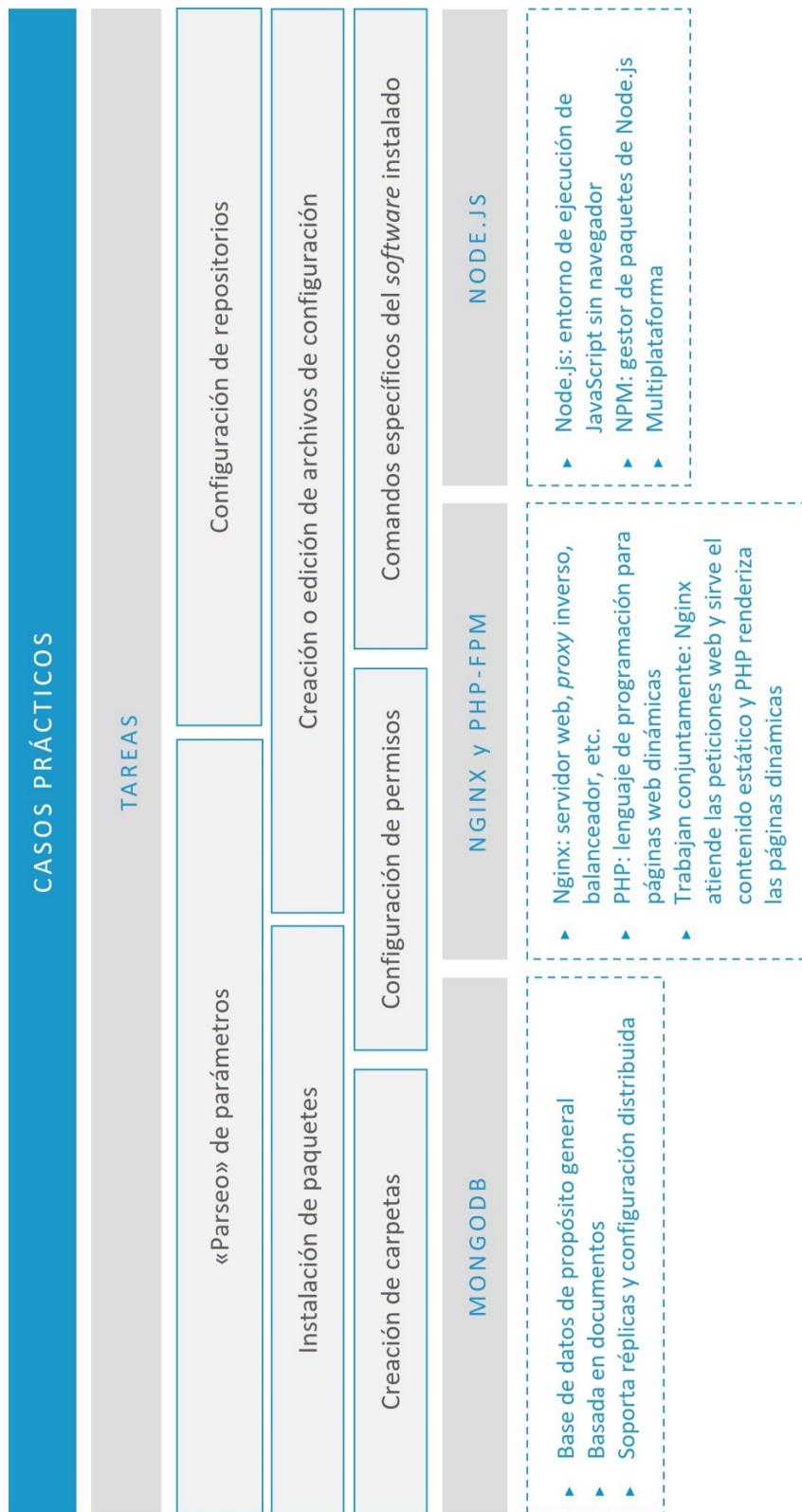
Administración de Sistemas en la Cloud

Automatización de instalación y configuración en Linux

Índice

Esquema	3
Ideas clave	4
4.1. Introducción y objetivos	4
4.2. Caso práctico: instalación de MongoDB	4
4.3. Caso práctico: instalación de PHP-FPM	16
4.4. Caso práctico: aplicación genérica de NodeJS	25
4.5. Caso práctico: automatización en AWS	31
4.6. Referencias bibliográficas	36

Esquema



4.1. Introducción y objetivos

Los conceptos de Bash son la primera piedra en el objetivo de automatizar las tareas administrativas en un servidor Linux. Cada comando y cada utilidad no hacen mucho por sí solos, pero, una vez combinados en un *script*, es posible automatizar cualquier instalación, configuración y despliegue necesario.

En este tema, se presentan ejemplos de *scripts* de instalación de **tres aplicaciones de servidor**: la base de datos MongoDB, el *framework* de páginas web dinámicas PHP-FPM y una aplicación genérica de NodeJS. En todos los casos, se analizará el *script* paso a paso. Un cuarto caso práctico demuestra cómo integrar un *script* de instalación en un despliegue automático de una instancia de la nube.

Los **objetivos** que se pretenden conseguir son:

- ▶ Tomar contacto con *scripts* complejos.
- ▶ Aprender a unir comandos de Bash y utilidades de línea de comandos para construir *scripts* avanzados.
- ▶ Familiarizarse con la documentación de aplicaciones de servidor.

4.2. Caso práctico: instalación de MongoDB

MongoDB es una base de datos de propósito general basada en [documentos](#). Soporta alta disponibilidad gracias a su funcionalidad de réplicas y consultas distribuidas en las instalaciones en modo *shard*. La instalación de un servidor *standalone* (en el sentido de que no estará conectado a otros servidores para replicar datos) es

relativamente sencilla y no hace falta más que seguir los pasos indicados en la documentación de [MongoDB](#). MongoDB ofrece una edición [Community](#) sin coste (y, por tanto, también sin soporte más allá de los foros).

A continuación, en el vídeo *Paso a paso de instalación MongoDB en Linux*, se verá una explicación y ejecución paso a paso del *script* de instalación y configuración de MongoDB en Linux.



Accede al vídeo

Script

Este *script* se basa en la documentación oficial de MongoDB. Usa diversas utilidades GNU y funciones de Bash para automatizar lo más posible la instalación. Tras el *script*, se detallan los apartados en los que está organizado.

```
#!/bin/bash

set -e

logger "Arrancando instalacion y configuracion de MongoDB"
USO="Uso : install.sh [opciones]
Ejemplo:
install.sh -u administrador -p password [-n 27017]
Opciones:
-u usuario
-p password
-n numero de puerto (opcional)
-a muestra esta ayuda
"

function ayuda() {
echo "${USO}"
if [[ ${1} ]]
```

```

then
    echo ${1}
fi
}

# Gestionar los argumentos
while getopts ":u:p:n:a" OPCION
do
    case ${OPCION} in
        u ) USUARIO=$OPTARG
            echo "Parametro USUARIO establecido con '${USUARIO}'";;
        p ) PASSWORD=$OPTARG
            echo "Parametro PASSWORD establecido";;
        n ) PUERTO_MONGOD=$OPTARG
            echo "Parametro PUERTO_MONGOD establecido con '${PUERTO_MONGOD}'";;
        a ) ayuda; exit 0;;
        : ) ayuda "Falta el parametro para -$OPTARG"; exit 1;; \?) ayuda "La
opcion no existe : $OPTARG"; exit 1;;
    esac
done

if [ -z ${USUARIO} ]
then
    ayuda "El usuario (-u) debe ser especificado"; exit 1
fi

if [ -z ${PASSWORD} ]
then
    ayuda "La password (-p) debe ser especificada"; exit 1
fi

if [ -z ${PUERTO_MONGOD} ]
then
    PUERTO_MONGOD=27017
fi

# Preparar el repositorio (apt-get) de mongodb añadir su clave apt
apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
4B7C549A058F8B6B

```

```

echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
xenial/mongodb-org/4.2 multiverse" | tee
/etc/apt/sources.list.d/mongodb.list

if [[ -z "$(mongo --version 2> /dev/null | grep '4.2.1')" ]]
then
    # Instalar paquetes comunes, servidor, shell, balanceador de shards y
    herramientas
    apt-get -y update \
    && apt-get install -y \
        mongodb-org=4.2.1 \
        mongodb-org-server=4.2.1 \
        mongodb-org-shell=4.2.1 \
        mongodb-org-mongos=4.2.1 \
        mongodb-org-tools=4.2.1 \
    && rm -rf /var/lib/apt/lists/* \
    && pkill -u mongodb || true \
    && pkill -f mongod || true \
    && rm -rf /var/lib/mongodb
fi

# Crear las carpetas de logs y datos con sus permisos
[[ -d "/datos/bd" ]] || mkdir -p -m 755 "/datos/bd"
[[ -d "/datos/log" ]] || mkdir -p -m 755 "/datos/log"

# Establecer el dueño y el grupo de las carpetas db y log
chown mongodb /datos/log /datos/bd
chgrp mongodb /datos/log /datos/bd

# Crear el archivo de configuración de mongodb con el puerto solicitado
mv /etc/mongod.conf /etc/mongod.conf.orig
(
cat <<MONGOD_CONF
# /etc/mongod.conf
systemLog:
destination: file
path: /datos/log/mongod.log
logAppend: true
storage:
dbPath: /datos/bd

```

```

engine: wiredTiger
journal:
  enabled: true
net:
  port: ${PUERTO_MONGOD}
security:
  authorization: enabled
MONGOD_CONF
) > /etc/mongod.conf

# Reiniciar el servicio de mongod para aplicar la nueva configuracion
systemctl restart mongod

logger "Esperando a que mongod responda..."
sleep 15

# Crear usuario con la password proporcionada como parametro
mongo admin << CREACION_DE_USUARIO
db.createUser({
  user: "${USUARIO}",
  pwd: "${PASSWORD}",
  roles:[{
    role: "root",
    db: "admin"
  },{
    role: "restore",
    db: "admin"
  }]
})
CREACION_DE_USUARIO

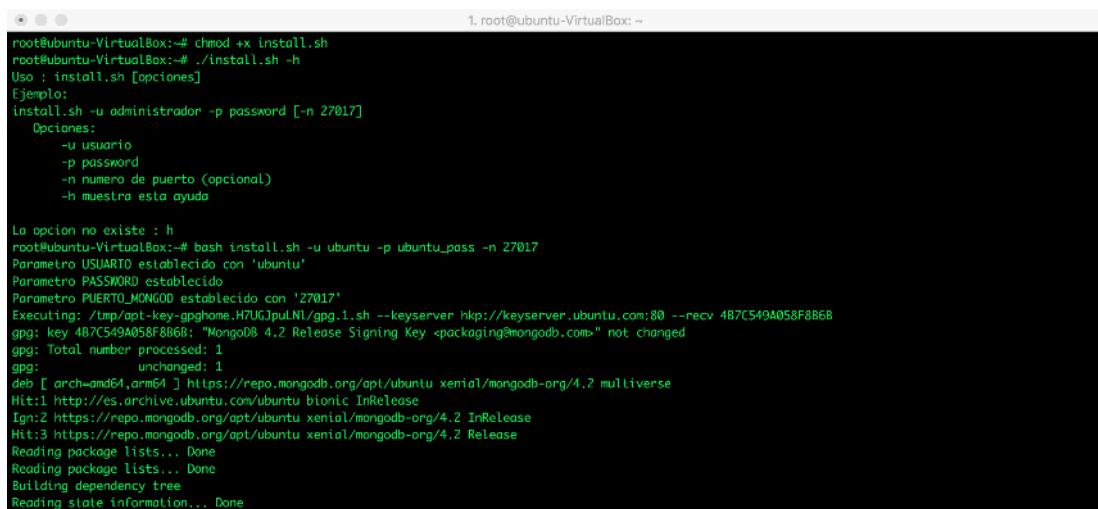
logger "El usuario ${USUARIO} ha sido creado con exito!"

exit 0

```

Ejecución

Tal como indican los comentarios al principio del *script*, los parámetros «usuario» y «password» son obligatorios, mientras que el «puerto» es opcional. En la Figura 1, se activa el *flag x* del *script*, para poder ejecutarlo directamente, y se invoca la ayuda. A continuación, se ejecuta con Bash (sería equivalente invocarlo directamente) con los parámetros de instalación: bash install.sh -u ubuntu -p ubuntu_pass -n 27017.



```
root@ubuntu-VirtualBox:~# chmod +x install.sh
root@ubuntu-VirtualBox:~# ./install.sh -h
Uso : install.sh [opciones]
Ejemplo:
install.sh -u administrador -p password [-n 27017]
Opciones:
-u usuario
-p password
-n numero de puerto (opcional)
-h muestra esta ayuda

La opcion no existe : h
root@ubuntu-VirtualBox:~# bash install.sh -u ubuntu -p ubuntu_pass -n 27017
Parametro USUARIO establecido con 'ubuntu'
Parametro PASSWORD establecido
Parametro PUERTO_MONGO establecido con '27017'
Executing: /tmp/tmp-key-gpphome.H7UGOpulNL/gpg_1.sh --keyserver hkp://keyserver.ubuntu.com:80 --recv 487C549A058F886B
gpg: key 487C549A058F886B: "MongoDB 4.2 Release Signing Key <packaging@mongodb.com>" not changed
gpg: Total number processed: 1
gpg: unchanged: 1
deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.2 multiverse
Hit:1 http://es.archive.ubuntu.com/ubuntu bionic InRelease
Ign:2 https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.2 InRelease
Hit:3 https://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/4.2 Release
Reading package lists... Done
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figura 1. Ejecución del *script* de instalación de MongoDB. Fuente: elaboración propia.

Shebang

El *script* comienza con el *shebang* para poder ejecutarlo directamente como un *script* de Bash.

```
#!/bin/bash
set -e
```

A continuación, se hace uso del comando **set** de Bash, que modifica comportamientos de la consola. Concretamente, **set -e** saldrá del *script* si cualquiera de los comandos falla. Este *script* es muy sencillo y las únicas situaciones de error que comprueba son los parámetros incorrectos. Si uno de los comandos falla, es

preferible no seguir adelante, ya que se ejecutarán comandos que asumen que las líneas anteriores han terminado con éxito.

Función de ayuda

El texto de ayuda se imprime con una función, en vez de con un simple echo, para poder imprimir un mensaje adicional si se ejecuta con un parámetro. Así, si se ejecuta `./install -h`, solo se imprime la ayuda, pero, si se introduce un parámetro que no existe, se imprime la ayuda seguida de un mensaje de error específico.

```
function ayuda() {  
    echo "${USO}"  
    if [[ ${1} ]]  
    then  
        echo ${1}  
    fi  
}
```

Captura de parámetros

El *script* no es una pieza de código fija que instala el *software* siempre de la misma manera. Los parámetros permiten personalizar la ejecución de un *script* para que sea útil en múltiples situaciones, sin necesidad de alterar el código. Al fin y al cabo, el *script* puede ser usado por administradores sin conocimientos avanzados de Bash y, aun así, ser igualmente útil.

Los parámetros están normalmente disponibles en las variables `$1`, `$2`, `$3`, etc. o en el array `$@`. El código para «parsear» los parámetros a partir de estas variables sería muy farragoso y esta tarea es tan habitual que hay utilidades que lo facilitan mucho. Este *script* usa el comando [getopts](#) de Bash: para cada elemento de `:u:p:n:h`, `getopts` guarda el valor del parámetro en la variable `OPCION`. Entonces, la expresión `case` facilita la ejecución de unas líneas de código para cada valor de `OPCION`. Es equivalente a una estructura `if-else`, pero más fácil de leer.

```

while getopts ":u:p:n:h" OPCION
do
    case ${OPCION} in
        u ) USUARIO=$OPTARG
            echo "Parametro USUARIO establecido con '${USUARIO}'";;
        p ) PASSWORD=$OPTARG
            echo "Parametro PASSWORD establecido";;
        n ) PUERTO_MONGOD=$OPTARG
            echo "Parametro PUERTO_MONGOD establecido con '${PUERTO_MONGOD}'";;
        h ) ayuda; exit 0;;
        : ) ayuda "Falta el parametro para -$OPTARG"; exit 1;;
        \?) ayuda "La opcion no existe : $OPTARG"; exit 1;;
    esac
done

```

Al terminar este bloque, los parámetros del *script* están disponibles en **variables** de Bash. Esto abre la puerta a simplificar el *script*, de manera que espere los parámetros directamente como variables de entorno. Este método obliga al usuario a definir las variables antes de invocar el *script*, pero puede ser útil según los requisitos del *script*, por ejemplo, un *script* principal puede invocar un segundo *script*: el primer *script* leería los parámetros y definiría las variables de entorno y el segundo no tendría más que leer las variables definidas del primero.

Validación

Siempre es necesario validar los datos de entrada, tanto si el *script* acepta parámetros como si espera los valores de entorno. La siguiente sección hace lo siguiente:

- ▶ Si el usuario o la contraseña no están definidos, imprime un mensaje de error y sale con un código de salida diferente de 0.
- ▶ Si el puerto no está definido, lo fija a un valor por defecto de 27017.

```

if [ -z ${USUARIO} ]
then
    ayuda "El usuario (-u) debe ser especificado"; exit 1

```

```

fi

if [ -z ${PASSWORD} ]
then
    ayuda "La password (-p) debe ser especificada"; exit 1
fi

if [ -z ${PUERTO_MONGOD} ]
then
    PUERTO_MONGOD=27017
fi

```

El comando [es un alias del comando test. Esta utilidad se usa para comprobar diferentes situaciones: si existe un fichero, si una variable de entorno que está definida, etc. De hecho, el *script* usa el *flag* -z, que comprueba si una cadena de texto tiene longitud cero (Robbins, 2010).

Repositorios e instalación

El siguiente bloque añade el repositorio de apt oficial de MongoDB para poder instalar la **última versión disponible** (4.2 en el momento de escribir esta sección). Las distribuciones suelen incorporar versiones anteriores (3.6 en el caso de [Ubuntu 18.04](#)).

```

# Preparar el repositorio (apt-get) de mongodb añadir su clave apt
apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv
4B7C549A058F8B6B
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
xenial/mongodb-org/4.2 multiverse" | tee
/etc/apt/sources.list.d/mongodb.list

```

A continuación, se instalan el servidor y las herramientas de línea de comandos, **solo si el servidor no está ya instalado**. La instalación es un proceso largo, así que es útil evitarlo si el *software* ya está instalado. Además, esta condición hace que el *script* sea **idempotente**. La idempotencia es una cualidad deseable en un *script*, por su

capacidad de recuperación automática: si el sistema falla durante la ejecución de un *script* idempotente por causas ajenas al *script*, una segunda ejecución puede llegar a finalizarse correctamente y, además, evitará ejecutar de nuevo procesos que han finalizado en la primera ejecución.

```
if [[ -z "$(mongo --version 2> /dev/null | grep '4.2.1')" ]]
then
    # Instalar paquetes comunes, servidor, shell, balanceador de shards y
    herramientas
    apt-get -y update \
    && apt-get install -y \
        mongodb-org=4.2.1 \
        mongodb-org-server=4.2.1 \
        mongodb-org-shell=4.2.1 \
        mongodb-org-mongos=4.2.1 \
        mongodb-org-tools=4.2.1 \
    && rm -rf /var/lib/apt/lists/* \
    && pkill -u mongodb || true \
    && pkill -f mongod || true \
    && rm -rf /var/lib/mongodb
fi
```

Configuración

El siguiente paso consiste en configurar los directorios que alojarán los ficheros de base de datos y de log. De nuevo, se hace uso del comando `test` para no crear las carpetas, si ya existen.

```
# Crear las carpetas de logs y datos con sus permisos
[[ -d "/datos/bd" ]] || mkdir -p -m 755 "/datos/bd"
[[ -d "/datos/log" ]] || mkdir -p -m 755 "/datos/log"

# Establecer el dueño y el grupo de las carpetas db y log
chown mongodb /datos/log /datos/bd
chgrp mongodb /datos/log /datos/bd
```

La configuración del servidor de MongoDB, al igual que muchos otros paquetes de software, consiste en un [fichero de configuración](#). El *script* incluye un [heredoc](#), es decir, el contenido del fichero de configuración, que puede contener un número arbitrario de líneas, está embebido en el propio *script*. El identificador MONGOD_CONF delimita el contenido del fichero y el comando cat se encarga de escribirlo en la ruta de destino. El contenido del fichero inserta valores de variables de entorno con \${PUERTO_MONGOD}.

```
(  
cat <<MONGOD_CONF  
# /etc/mongod.conf  
systemLog:  
    destination: file  
    path: /datos/log/mongod.log  
    logAppend: true  
storage:  
    dbPath: /datos/bd  
    engine: wiredTiger  
    journal:  
        enabled: true  
net:  
    port: ${PUERTO_MONGOD}  
security:  
    authorization: enabled  
MONGOD_CONF  
) > /etc/mongod.conf
```

Creación del usuario por defecto

Una instalación nueva no incluye ningún usuario, así que el *script* se encarga de crear uno a partir de los parámetros proporcionados por el usuario. Para esto, se hace uso de la consola de MongoDB, mongo. Aparte de ofrecer una línea de comandos interactiva, mongo acepta comandos nativos. Para esto, se usa otro HEREDOC, que en este caso se inyecta en el comando mongo a través de la entrada estándar.

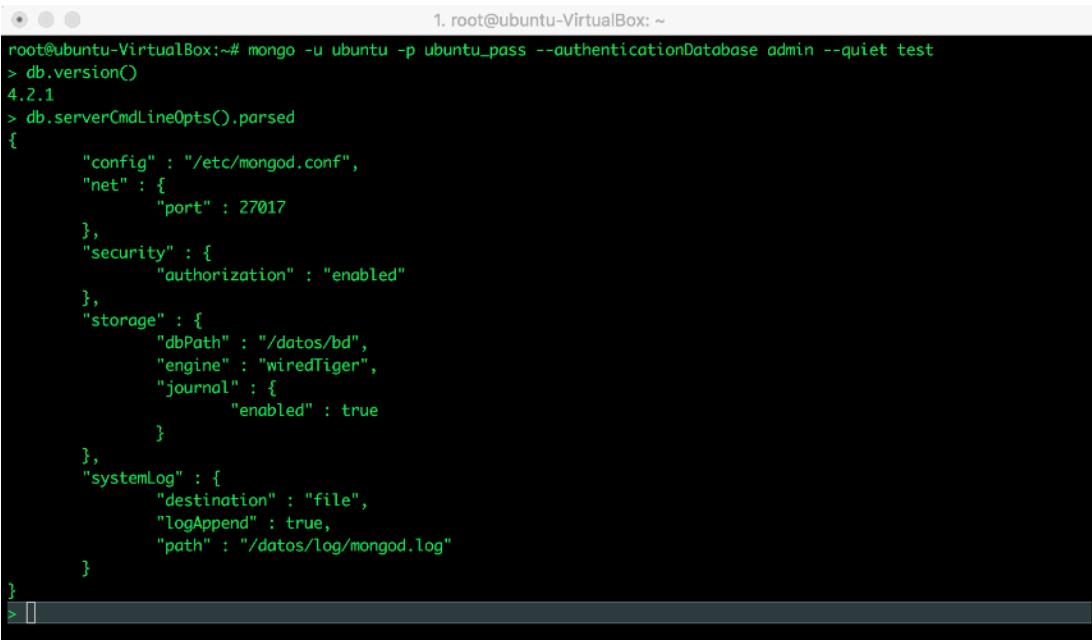
```

# Crear usuario con la password proporcionada como parametro
mongo admin << CREACION_DE_USUARIO
db.createUser({
    user: "${USUARIO}",
    pwd: "${PASSWORD}",
    roles:[{
        role: "root",
        db: "admin"
    },{
        role: "restore",
        db: "admin"
    }]
})
CREACION_DE_USUARIO

```

Comprobación

Si todo ha ido bien, el *script* ejecuta `exit 0` para notificar que ha terminado con éxito. La manera de comprobar que el servidor esta correctamente configurado sería iniciar sesión con los datos del usuario, tal como muestra la Figura 2.



```

1. root@ubuntu-VirtualBox: ~
root@ubuntu-VirtualBox:~# mongo -u ubuntu -p ubuntu_pass --authenticationDatabase admin --quiet test
> db.version()
4.2.1
> db.serverCmdLineOpts().parsed
{
  "config" : "/etc/mongod.conf",
  "net" : {
    "port" : 27017
  },
  "security" : {
    "authorization" : "enabled"
  },
  "storage" : {
    "dbPath" : "/datos/bd",
    "engine" : "wiredTiger",
    "journal" : {
      "enabled" : true
    }
  },
  "systemLog" : {
    "destination" : "file",
    "logAppend" : true,
    "path" : "/datos/log/mongod.log"
  }
}
> []

```

Figura 2. Inicio de sesión en MongoDB tras la ejecución del *script*. Fuente: elaboración propia.

4.3. Caso práctico: instalación de PHP-FPM

Este segundo *script* de ejemplo instalará un servidor PHP sobre Nginx. **PHP** es un lenguaje interpretado muy extendido en el desarrollo de aplicaciones web dinámicas. Algunos de los administradores de contenidos basados en PHP son Wordpress, Drupal y Moodle. Se puede instalar como un servidor web propio o junto a un servidor web específico, como Apache, IIS o [Nginx](#).

Nginx es un servidor web que también puede usarse como *proxy* inverso, balanceador de nivel 5 y caché de HTTP (*hypertext transfer protocol*) (Nginx, s. f.). Permite servir, por ejemplo, contenido estático por sí mismo y contenido dinámico mediante una integración con otro *software*. Nginx Inc., la compañía detrás de Nginx, ofrece una versión de Nginx de pago con más funcionalidad. Este *script* instalará la versión gratuita.

El *script* de PHP es similar en estructura al *script* de instalación de MongoDB, pero incluye algunas novedades, como la ejecución condicionada a la plataforma y la edición de fichero de configuración al vuelo.

```
#!/bin/bash

set -e

logger "Arrancando instalacion y configuracion de PHP-FPM"
USO="Uso : install.sh [opciones]
Ejemplo:
install.sh -s mysite.com

Opciones:
    -s nombre del site en nginx
    -a muestra esta ayuda
"

function ayuda() {
```

```

echo "${USO}"
if [[ ${1} ]]
then
    echo ${1}
fi
}

# Gestionar las opciones
while getopts ":u:g:l:a" OPCION
do
    case ${OPCION} in
        s ) NGINX_SITE=$OPTARG
            logger "Parametro NGINX_SITE establecido con '${NGINX_SITE}'"
            ;;
        a ) ayuda; exit 0;;
        : ) ayuda "Falta el parámetro para -$OPTARG"; exit 1;;
        \?) ayuda "La opción no existe : $OPTARG"; exit 1;;
    esac
done

if [ -z ${NGINX_SITE} ]
then
    NGINX_SITE="php.local"
fi

# Instalar PHP-FPM usando apt-get o yum dependiendo de la plataforma
if [[ -e /etc/redhat-release || -e /etc/system-release ]]
then

    # Solo se soporta la instalacion en la release 7 de RedHat y CentOS
    OS=$(rpm -q --whatprovides redhat-release | cut -d"-" -f1)
    RELEASE=$(rpm -q --whatprovides redhat-release | cut -d"-" -f3)

    if [[ "${OS}" != "centos" && "${OS}" != "redhat" ]]
    then
        logger "La instalacion solo esta soportada en RedHat y CentOS"
        exit 2
    fi

    if [[ "${RELEASE}" != "7" ]]

```

```

then
    logger "La instalacion solo esta soportada en CentOS y RedHat release
7"
    exit 3
fi

# Configuracion de SELinux
setenforce permissive

# Preparacion de yum con los repositorios de nginx y php-fpm
(
cat << NGINX_REPO
[nginx]
name=nginx repo
baseurl=http://nginx.org/packages/mainline/rhel/7/\$basearch/
gpgcheck=0
enabled=1
NGINX_REPO
) > /etc/yum.repos.d/nginx.repo

yum install -y epel-release
yum install -y http://rpms.remirepo.net/enterprise/remi-release-7.rpm
yum install -y yum-utils
yum-config-manager --enable remi-php72
yum update -y

# Instalacion y configuracion de nginx
yum install -y nginx

(
cat << NGINX_CONF
server {
    listen      80;
    root /var/www/html;
    index index.php index.html index.htm;
    server_name $NGINX_SITE;

    location / {
        try_files \$uri \$uri/ =404;
    }
)

```

```

location ~ \.php$ {
    fastcgi_pass    unix:/var/run/php/php7.2-fpm.sock;
    fastcgi_index   index.php;
    fastcgi_param           SCRIPT_FILENAME
                           $document_root$fastcgi_script_name;
    include          fastcgi_params;
}

}

NGINX_CONF
) > /etc/nginx/conf.d/$NGINX_SITE.conf

rm /etc/nginx/conf.d/default.conf

systemctl start nginx

# Instalacion y configuracion de php-fpm
yum install -y php72 php72-php-fpm php72-php-gd \
    php72-php-json php72-php-mbstring php72-php-mysqlnd \
    php72-php-xml php72-php-xmlrpc php72-php-opcache

mkdir /var/run/php
chown nginx:nginx /var/run/php/

sed -i "s/user =.*user = nginx/g" /etc/opt/remi/php72/php-
fpm.d/www.conf
sed -i "s/group =.*group = nginx/g" /etc/opt/remi/php72/php-
fpm.d/www.conf
sed -i "s/listen =.*listen = \/var\/run\/php\/php7.2-fpm.sock/g"
/etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/.listen.owner =.*listen.owner = nginx/g"
/etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/.listen.group =.*listen.group = nginx/g"
/etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/.listen.mode =.*listen.mode = 0660/g"
/etc/opt/remi/php72/php-fpm.d/www.conf

systemctl enable php72-php-fpm.service
systemctl start php72-php-fpm.service
# Limpieza de archivos temporales tras la instalacion

```

```

yum clean all

else

# configuracion de repositorios
export DEBIAN_FRONTEND=noninteractive
apt-get install software-properties-common
add-apt-repository -y ppa:ondrej/nginx-mainline
apt-get install -y nginx php7.2 php7.2-fpm

# Crear el archivo del site de nginx
(
cat << NGINX_CONF
server {
    listen 80;
    root /var/www/html;
    index index.php index.html index.htm;
    server_name $NGINX_SITE;

    location / {
        try_files \$uri \$uri/ =404;
    }

    location ~ \.php$ {
        include snippets/fastcgi-php.conf;
        fastcgi_pass unix:/var/run/php/php7.2-fpm.sock;
    }
}
NGINX_CONF
) > /etc/nginx/sites-available/$NGINX_SITE

rm /etc/nginx/sites-enabled/default
sudo ln -s /etc/nginx/sites-available/$NGINX_SITE /etc/nginx/sites-enabled/$NGINX_SITE

systemctl restart nginx

# La configuracion por defecto de PHP-FPM es suficiente
fi

```

```

# Creacion de un fichero de prueba en la raiz
# Sera accesible tras la instalacion en http://<ip_del_servidor>/info.php
mkdir -p /var/www/html
echo "<?php phpinfo(); ?>" > /var/www/html/info.php
id -u nginx && chown -R nginx:nginx /var/www || chown -R www-data:www-data
/var/www
chmod -R 755 /var/www

logger "Instalacion completada con exito; visite http://<ip>/info.php para
comprobarlo"

exit 0

```

Detección de plataforma

Aunque el núcleo de Linux pueda ser el mismo, las diferentes distribuciones en la industria siguen diferentes enfoques en cuanto a la instalación de paquetes, configuración del sistema, etc. Incluso dentro de una misma distribución, dos versiones pueden diferir tanto como dos distribuciones diferentes. Por tanto, es habitual que los *scripts* de instalación sean capaces de **detectar en qué plataforma se están ejecutando**. El código puede estar preparado para ejecutarse en diferentes plataformas o terminar con error si se ejecuta en una plataforma para la que no está preparada.

Este *script* incluye ejemplos de ambos modelos: por un lado, detecta si se está ejecutando en una distribución basada en RedHat. En tal caso, termina con error si no se trata de un RedHat o CentOS versión 7. Si no es una distribución basada en RedHat, el *script* asume que se está ejecutando en una distribución Ubuntu 18.

```

# Instalar PHP-FPM usando apt-get o yum dependiendo de la plataforma
if [[ -e /etc/redhat-release || -e /etc/system-release ]]
then

    # Solo se soporta la instalacion en la release 7 de RedHat y CentOS
    OS=$(rpm -q --whatprovides redhat-release | cut -d"-" -f1)

```

```

RELEASE=$(rpm -q --whatprovides redhat-release | cut -d"-" -f3)

if [[ "${OS}" != "centos" && "${OS}" != "redhat" ]]
then
    logger "La instalacion solo esta soportada en RedHat y CentOS"
    exit 2
fi

if [[ "${RELEASE}" != "7" ]]
then
    logger "La instalacion solo esta soportada en CentOS y RedHat release
7"
    exit 3
fi

# ... prosigue la instalación en RedHat y CentOS

else

    # ... prosigue la instalación en Ubuntu

fi

```

Instalación en CentOS

El **esquema de la instalación** en CentOS no difiere mucho de la instalación en Ubuntu: configurar repositorios, instalación de paquetes, configurar y arrancar servicios. La principal diferencia es que el gestor de paquetes es yum en vez de apt-get. CentOS incluye una configuración por defecto de [SELinux](#) demasiado restrictiva para el funcionamiento de PHP en conjunción con Nginx, así que hay que incluir el comando `setenforce permissive`.

Finalmente, aunque las versiones de Nginx y PHP-FPM son las mismas, los paquetes de instalación de cada plataforma tienen configuraciones por defecto diferentes. Por ejemplo, la configuración de PHP-FPM en Ubuntu no requiere modificaciones, pero la configuración en CentOS requiere reconfigurar el *socket* Unix y los usuarios con los

que se ejecuta el proceso. En vez de escribir el fichero de configuración entero, el comando [sed](#) edita el fichero al vuelo, editando solo las líneas necesarias.

```
sed -i "s/user =.*user = nginx/g" /etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/group =.*group = nginx/g" /etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/listen =.*listen = \var\run\php\php7.2-fpm.sock/g" /etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/.*listen.owner =.*listen.owner = nginx/g" /etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/.*listen.group =.*listen.group = nginx/g" /etc/opt/remi/php72/php-fpm.d/www.conf
sed -i "s/.*listen.mode =.*listen.mode = 0660/g" /etc/opt/remi/php72/php-fpm.d/www.conf
```

El proceso Nginx se ejecuta con el usuario `nginx` por defecto en CentOS, pero con el usuario `www-data` en Ubuntu. Ya que la creación del fichero de ejemplo está fuera de la condición en función de la plataforma, se usa una expresión `if-else` en la misma línea: si el comando `id -u nginx` termina satisfactoriamente, se ejecuta el comando a continuación de `&&`. Si termina en error, se ejecuta el comando a continuación de `||`. El comando `id` devuelve el identificador del nombre de usuario y falla si el usuario no existe.

```
id -u nginx && chown -R nginx:nginx /var/www || chown -R www-data:www-data /var/www
```

Comprobación

El fichero `info.php`, creado por el *script*, contiene la función `phpinfo()` de PHP, que imprime una larga lista de opciones de instalación y módulos en ejecución. Si se recupera el fichero con un navegador y se obtiene un fichero de texto plano con el contenido `<?php phpinfo(); ?>`, la instalación no ha funcionado. Sin embargo, si se obtiene una página como la de la Figura 3, la instalación terminó con éxito.

PHP Version 7.2.3-1ubuntu1	
System	Linux ubuntu-VirtualBox 4.15.0-29-generic #31-Ubuntu SMP Tue Jul 17 15:39:52 UTC 2018 x86_64
Build Date	Mar 14 2018 22:03:58
Server API	FPM/FastCGI
Virtual Directory Support	disabled
Configuration File (php.ini) Path	/etc/php/7.2/fpm
Loaded Configuration File	/etc/php/7.2/fpm/php.ini
Scan this dir for additional .ini files	/etc/php/7.2/fpm/conf.d
Additional .ini files parsed	/etc/php/7.2/fpm/conf.d/10-opcache.ini, /etc/php/7.2/fpm/conf.d/10-pdo.ini, /etc/php/7.2/fpm/conf.d/20-calendar.ini, /etc/php/7.2/fpm/conf.d/20-ctype.ini, /etc/php/7.2/fpm/conf.d/20-exif.ini, /etc/php/7.2/fpm/conf.d/20-fileinfo.ini, /etc/php/7.2/fpm/conf.d/20-ftp.ini, /etc/php/7.2/fpm/conf.d/20-gettext.ini, /etc/php/7.2/fpm/conf.d/20-iconv.ini, /etc/php/7.2/fpm/conf.d/20-json.ini, /etc/php/7.2/fpm/conf.d/20-phar.ini, /etc/php/7.2/fpm/conf.d/20-posix.ini, /etc/php/7.2/fpm/conf.d/20-readline.ini, /etc/php/7.2/fpm/conf.d/20-shmop.ini, /etc/php/7.2/fpm/conf.d/20-sockets.ini, /etc/php/7.2/fpm/conf.d/20-sysvmsg.ini, /etc/php/7.2/fpm/conf.d/20-sysvsem.ini, /etc/php/7.2/fpm/conf.d/20-sysvshm.ini, /etc/php/7.2/fpm/conf.d/20-tokenizer.ini
PHP API	20170718
PHP Extension	20170718
Zend Extension	320170718
Zend Extension Build	API320170718,NTS
PHP Extension Build	API20170718,NTS
Debug Build	no
Thread Safety	disabled
Zend Signal Handling	enabled
Zend Memory Manager	enabled
Zend Multibyte Support	disabled
IPv6 Support	enabled
DTrace Support	available, disabled

Figura 3. Página de Phpinfo tras la instalación. Fuente: elaboración propia.

También es posible comprobar la instalación localmente en la misma línea de comandos, usando wget, como en la **¡Error! No se encuentra el origen de la referencia. 4.**

```
root@ubuntu-VirtualBox:~# wget -O - http://localhost/info.php | head
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"><head>
<style type="text/css">
body {background-color: #fff; color: #222; font-family: sans-serif;}
pre {margin: 0; font-family: monospace;}
a:link {color: #009; text-decoration: none; background-color: #fff;}
a:hover {text-decoration: underline;}
table {border-collapse: collapse; border: 0; width: 934px; box-shadow: 1px 2px 3px #ccc;}
.center {text-align: center;}
.center table {margin: 1em auto; text-align: left;}
Cannot write to '-' (Broken pipe).
root@ubuntu-VirtualBox:~#
```

Figura 4. Comando wget recuperando la página de Phpinfo. Fuente: elaboración propia.

4.4. Caso práctico: aplicación genérica de NodeJS

[NodeJS](#) es un **entorno de ejecución** de JavaScript multiplataforma y de **código libre**. Aunque JavaScript nació como un lenguaje de *scripting* para navegadores web, NodeJS ejecuta el código fuera del navegador. Los desarrolladores pueden usarlo para programar herramientas de línea de comandos y herramientas de servidor. Este último caso es muy habitual: hay *frameworks* como [ExpressJS](#), también conocido como Express, que facilitan el desarrollo de servidores web dinámicos al estilo de PHP.

[NPM](#) (*nearly picked makefiles*) es un gestor de paquetes JavaScript por defecto de NodeJS. Además de una herramienta de línea de comandos pensada para gestionar las dependencias de las aplicaciones de NodeJS, el ecosistema de NPM tiene a su disposición un repositorio público de paquetes, similar en su naturaleza a los repositorios de apt-get o yum. NPM se integra con NodeJS de tal modo que los desarrolladores definen las dependencias de la aplicación en el fichero package.json y NPM las instala y las expone al código local al arrancar la aplicación.

Este *script* instalará no solo NodeJS y NPM, sino que está preparado para descargar el código de una aplicación desde un repositorio de Github.

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

```
AYUDA="Las variables APP_ENTRY_POINT, APP_FOLDER y CLONE_URL deben tener valores validos."
```

Ejemplo:

```
export APP_ENTRY_POINT=bin/www  
export APP_FOLDER=/opt/app  
export CLONE_URL=https://github.com/plesk/node-express  
export BRANCH=master  
"  
"
```

```
# verificacion de parametros
```

```

if [[ -z ${APP_ENTRY_POINT} || -z ${APP_FOLDER} || -z ${CLONE_URL} ]]; ;
then
    echo "${AYUDA}"
    exit 1
fi

if [[ -z ${BRANCH} ]]
then
    BRANCH=master
fi

echo "Arrancando instalacion de $APP_ENTRY_POINT@$BRANCH en $APP_FOLDER"

# configuracion para githubb por SSH
mkdir -p ~/.ssh
if [ ! -f ~/.ssh/config ]; then

    cat > ~/.ssh/config << SSH_CONFIG
Host github.com
    StrictHostKeyChecking no
    UserKnownHostsFile=/dev/null
SSH_CONFIG

fi

# instalacion de curl y git en funcion de la distribucion
if [[ -e /etc/redhat-release || -e /etc/system-release ]]; then
    yum -y check-update
    yum install git curl -y
else
    apt-get -y update
    apt-get install git curl -y
fi

# metodo alternativo de deteccion de distribucion
# instalacion de nodeJS y npm
if [[ -x /usr/bin/yum ]]; then
    OS=$(rpm -q --whatprovides redhat-release | cut -d"-" -f1)

    case $OS in

```

```

        system)
            yum -y install openssl-devel nodejs npm --enablerepo=epel
            ;;
        *)
            yum -y install epel-release
            yum -y install openssl-devel nodejs npm
            ;;
    esac
else
    if [[ ! -z $(uname -a | grep -i ubuntu) ]]; then curl -sL
https://deb.nodesource.com/setup_12.x | sudo bash -; fi
    apt-get -y install nodejs
fi

# herramienta para ejecutar una script indefinidamente
# la aplicacion no sera un servicio, pero se comportara como tal
npm install forever -g

# descarga del codigo del repositorio
[[ -z "$APP_FOLDER" ]] && TARGET_DIRECTORY=$(basename "$CLONE_URL" .git) ||
TARGET_DIRECTORY=$APP_FOLDER

if [ ! -d $TARGET_DIRECTORY ]; then
    # clonado inicial
    if [ -z '$BRANCH' ]; then
        git clone $CLONE_URL $APP_FOLDER
    else
        git clone -b $BRANCH $CLONE_URL $TARGET_DIRECTORY
    fi
else
    # actualización de nuevos commits
    cd $TARGET_DIRECTORY
    git remote set-url origin {{ CLONE_URL }}
    git clean -f
    git fetch --all
    git fetch --tags
    if [ -z '$BRANCH' ]; then
        git reset --hard origin
    else
        git reset --hard "origin/$BRANCH"
    fi
fi

```

```

    fi

    git pull origin $BRANCH

fi

# instalacion de las dependencias de nodeJS
pushd $APP_FOLDER
npm install
popd

# parada de los procesos de forever
PATH="$PATH:/usr/local/sbin:/usr/local/bin"
ps aux | grep $APP_FOLDER/$APP_ENTRY_POINT | grep -v grep -q
if [ $? -eq 0 ]; then
    forever stopall
fi

# arranque del script de entrada de la aplicacion
forever --minUptime 1000 --spinSleepTime 1000 \
    start $APP_FOLDER/$APP_ENTRY_POINT

```

Parámetros

Al contrario de los otros *scripts*, este no acepta parámetros por la línea de comandos, sino que espera que el usuario los haya fijado como **variables de entorno**. Por tanto, no hay «parseo» de comandos, pero sí comprobación de estos. Uno de los parámetros, **BRANCH**, es opcional y toma el valor por defecto de **master**.

```

# verificacion de parametros
if [[ -z ${APP_ENTRY_POINT} || -z ${APP_FOLDER} || -z ${CLONE_URL} ]] ;
then
    echo "${AYUDA}"
    exit 1
fi

if [[ -z ${BRANCH} ]]
then
    BRANCH=master

```

```
fi
```

Instalación multidistribución

El *script* está preparado para instalar la aplicación, tanto en distribuciones RedHat como Ubuntu. En la primera instalación de Git y Curl, el *script* hace uso de una comprobación similar a la vista en los *scripts* anteriores:

```
if [[ -e /etc/redhat-release || -e /etc/system-release ]]; then
```

Más adelante, sin embargo, aprovecha otra diferencia entre las distribuciones: la existencia del fichero /usr/bin/yum.

```
if [[ -x /usr/bin/yum ]]; then
```

Descarga de la aplicación y dependencias

Una vez que NodeJS y NPM están instalados, el resto del *script* no depende de la distribución. En primer lugar, NPM instala el paquete forever, que se puede usar para configurar aplicaciones de NodeJS como si fueran servicios.

```
npm install forever -g
```

A continuación, el *script* descarga la aplicación desde el repositorio de Github. Hay dos opciones: si la aplicación no existe, el *script* clona el repositorio de cero, pero, si ya ha sido instalada, actualiza la rama seleccionada al último *commit*.

```
# descarga del código del repositorio
[[ -z "$APP_FOLDER" ]] && TARGET_DIRECTORY=$(basename "$CLONE_URL" .git) ||
TARGET_DIRECTORY=$APP_FOLDER

if [ ! -d $TARGET_DIRECTORY ]; then
    # clonado inicial
    if [ -z '$BRANCH' ]; then
```

```

        git clone $CLONE_URL $APP_FOLDER
    else
        git clone -b $BRANCH $CLONE_URL $TARGET_DIRECTORY
    fi
else
    # actualización de nuevos commits
    cd $TARGET_DIRECTORY
    git remote set-url origin {{ CLONE_URL }}
    git clean -f
    git fetch --all
    git fetch --tags
    if [ -z '$BRANCH' ]; then
        git reset --hard origin
    else
        git reset --hard "origin/$BRANCH"
    fi

    git pull origin $BRANCH
fi

```

La instalación de las dependencias de NodeJS se hace a continuación con NPM. Los comandos pushd y popd sirven para cambiar el directorio de trabajo a uno diferente, para luego volver al original.

```

# instalacion de las dependencias de nodeJS
pushd $APP_FOLDER
npm install
popd

```

Dado que la aplicación puede estar en ejecución, el *script* detiene cualquier ejecución de procesos de forever, siempre y cuando la aplicación esté en ejecución.

```

# parada de los procesos de forever
PATH="$PATH:/usr/local/sbin:/usr/local/bin"
ps aux | grep $APP_FOLDER/$APP_ENTRY_POINT | grep -v grep -q
if [ $? -eq 0 ]; then
    forever stopall
fi

```

Finalmente, el *script* hace uso de forever para arrancar la aplicación. En este punto, es posible verificar si la aplicación ha arrancado satisfactoriamente. Por ejemplo, si se instala la aplicación de prueba de [Github](#), la configuración por defecto arranca el servidor HTTP en el puerto 3000, por lo que una simple vista con un navegador puede servir de verificación.

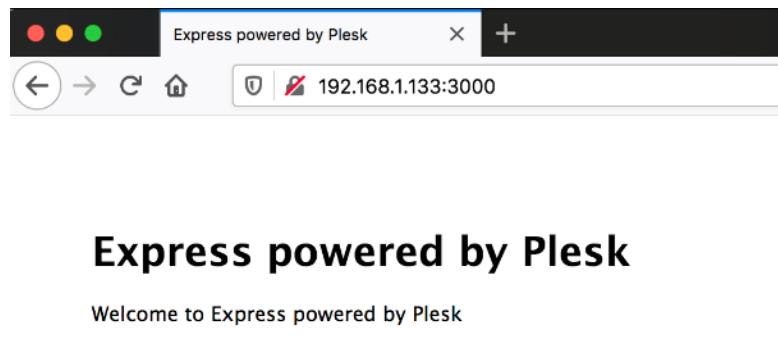


Figura 5. Comprobación de la aplicación NodeJS. Fuente: elaboración propia.

4.5. Caso práctico: automatización en AWS

Los *scripts* de los ejemplos anteriores se han demostrado en una ejecución interactiva, pero nada impide que se ejecuten desde una **herramienta de automatización externa**. Por ejemplo, Ansible o Puppet podrían usarlos, aunque en esos casos sería recomendable usar las directivas propias de las herramientas para aprovechar su potencial.

En este caso práctico, se va a hacer uso del *user data* de las instancias de AWS (Amazon Web Services) EC2 (AWS, s. f.). Esta característica es un trozo de código que se puede usar para automatizar tareas y ejecutar *scripts* una vez que arranca la instancia. Esto significa que no es necesario siquiera iniciar sesión en el sistema operativo de la instancia para ejecutar los comandos. Si la instancia se despliega como parte de un grupo de autoescalado o como respuesta a una llamada de API

(*application programming interface*), tanto el proceso de despliegue como el de configuración de la aplicación se habrán llevado a cabo sin intervención humana.

Las instancias Linux de EC2 aceptan dos tipos de *user data*: *scripts de shell* y directivas de Cloud-init. Este caso práctico demostrará ambos métodos, con la instalación de un servidor web con la pila LAMP (Linux, Apache, MySQL y PHP) en una instancia de tipo Amazon Linux 2.

User data con *scripts de shell*

El *script* que se va a usar como ejemplo es el siguiente:

```
#!/bin/bash
yum update -y
amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2
yum install -y httpd mariadb-server
systemctl start httpd
systemctl enable httpd
usermod -a -G apache ec2-user
chown -R ec2-user:apache /var/www
chmod 2775 /var/www
find /var/www -type d -exec chmod 2775 {} \;
find /var/www -type f -exec chmod 0664 {} \;
echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php
```

Es mucho más sencillo que los anteriores por dos razones: la distribución es conocida, por lo que no hay que considerar diferentes gestores de paquetes, y el *script* no acepta parámetros de usuario. Aunque se han obviado algunas comprobaciones de error para simplificarlo, el *script* es totalmente funcional.

Añadir estas tareas durante el arranque de la instancia incrementa el tiempo que esta tarda en estar disponible. En un despliegue manual, es necesario tener esto en cuenta antes de poder lanzar pruebas.

Los *scripts* de *shell* de *user data* deben empezar con el *shebang*, `#!/bin/bash` (o la *shell* necesaria, si procede). Estos *scripts*, además, se ejecutan como root y en modo no interactivo, por lo que:

- ▶ No es necesario ejecutar ninguno de los comandos con Sudo.
- ▶ Los comandos no pueden esperar entrada de datos del usuario.

El *log* de salida `/var/log/cloud-init-output.log` contendrá la salida estándar del *script*. En caso de error, este *log* se puede usar para verificar la salida de los comandos. Además, el *script* se copia a la ruta `/var/lib/cloud/instances/<instance-id>/` al procesarlo, por lo que es posible comprobar que el *script* que se ha ejecutado es efectivamente el deseado.

En un despliegue en la consola web, el *user data* se introduce en el paso de selección de los detalles de la instancia. Se puede seleccionar un fichero con el *script* o introducir el contenido del *script* directamente, tal como se indica en la Figura 6.

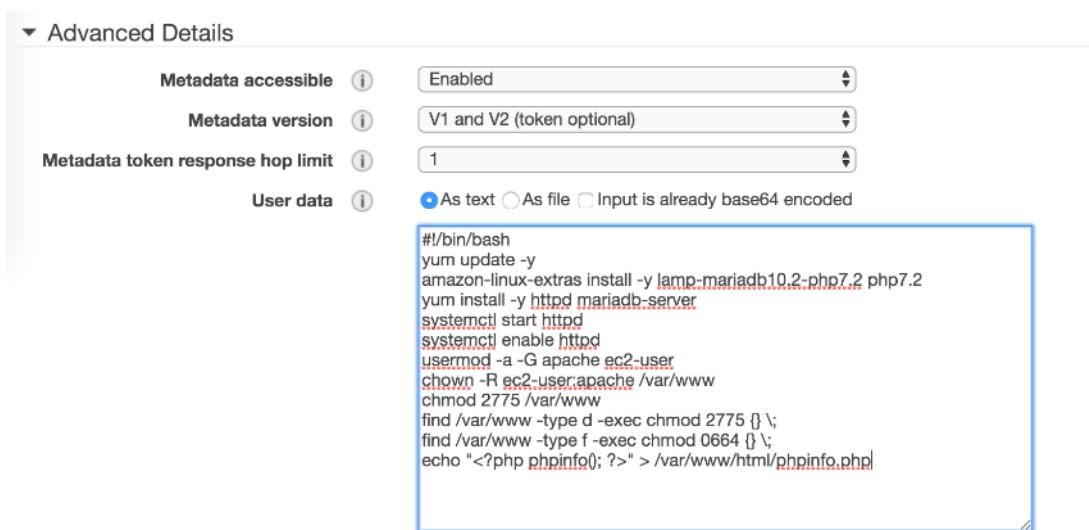


Figura 6. *User data* en la consola web. Fuente: elaboración propia.

El grupo de seguridad deberá permitir el acceso por HTTP, ya que la instancia va a alojar un servidor web.

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	My IP 139.47.100.40/32	e.g. SSH for Admin
HTTP	TCP	80	Custom 0.0.0.0/0, ::/0	e.g. SSH for Admin

Figura 7. Grupo de seguridad para servidor web. Fuente: elaboración propia.

Una vez arrancada la instancia, la vista de detalles contiene tanto la IP pública como un nombre DNS (*domain name system*) también público. Este nombre podrá usarse para acceder a la página de prueba `info.php` creada por el *script*.

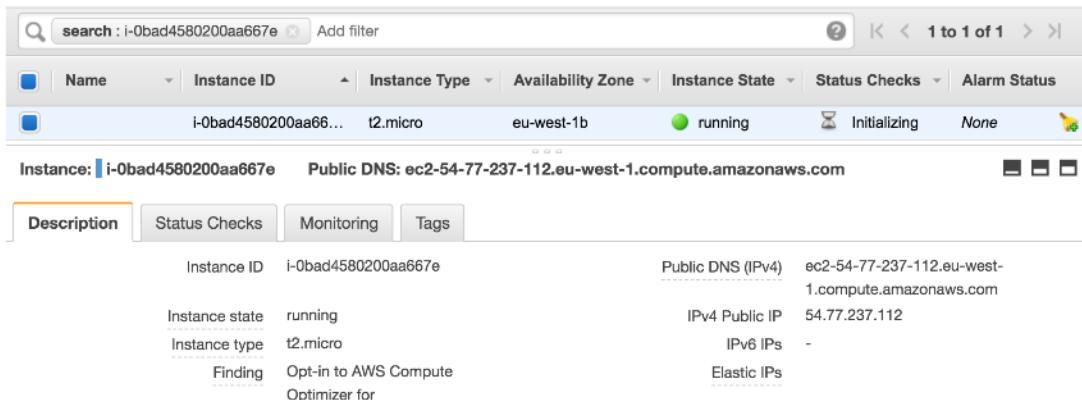


Figura 8. Instancia arrancada. Fuente: elaboración propia.

Figura 9. Página de prueba cargada desde el DNS público de la instancia. Fuente: elaboración propia.

User data con directivas de Cloud-init

[Cloud-init](#) es un estándar de facto de la industria para la inicialización de instancias de nube multiplataforma. Está soportado por la gran mayoría de proveedores de nube pública y plataformas de nube privada. AWS hace uso de Cloud-init activamente para configurar las claves públicas de inicio de sesión en el fichero `~/.ssh/authorized_keys` del usuario `ec2-user`. En el ejemplo anterior, aunque el usuario proporciona un *script* de Bash directamente, el *user data* hace uso de Cloud-init para ejecutarlo.

AWS soporta el uso de directivas de Cloud-init directamente. Esto da más control al administrador, que puede establecer más parámetros de configuración, no solo el *script* a ejecutar. AWS identifica que el tipo de contenido del *user data* es Cloud-init cuando la primera línea es `#cloud-config`. Las siguientes líneas consiguen el mismo objetivo de instalación del servidor web.

```
#cloud-config
repo_update: true
repo_upgrade: all

packages:
  - httpd
  - mariadb-server

runcmd:
  - [ sh, -c, "amazon-linux-extras install -y lamp-mariadb10.2-php7.2 php7.2" ]
  - systemctl start httpd
  - sudo systemctl enable httpd
  - [ sh, -c, "usermod -a -G apache ec2-user" ]
  - [ sh, -c, "chown -R ec2-user:apache /var/www" ]
  - chmod 2775 /var/www
  - [ find, /var/www, -type, d, -exec, chmod, 2775, {}, \; ]
  - [ find, /var/www, -type, f, -exec, chmod, 0664, {}, \; ]
  - [ sh, -c, 'echo "<?php phpinfo(); ?>" > /var/www/html/phpinfo.php' ]
```

El último bloque es muy parecido al *script* de Bash, con la diferencia de que cada línea pasa a ser un *array* de parámetros. La principal diferencia es que los paquetes de `httpd` y `mariadb-server` no se instalan con `yum`, sino a través de la directiva `packages`, específica de [Cloud-init](#). Otras directivas permiten la creación de usuarios, creación de ficheros de configuración, configuración de certificados de autoridades de confianza, etc.

El despliegue de la instancia es idéntico al ejemplo anterior, salvo que el contenido de *user data* contiene la directiva de Cloud-init, tal como muestra la Figura 10.

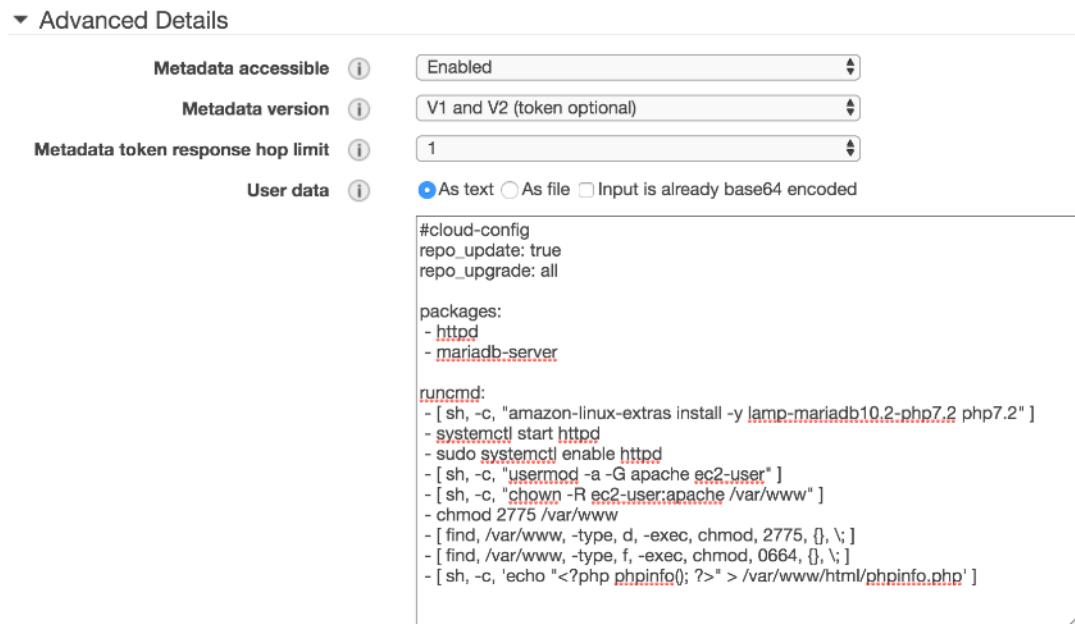


Figura 10. *User data* con *script* de Cloud-init. Fuente: elaboración propia.

4.6. Referencias bibliográficas

AWS. (s. f.). *Run commands on your Linux instance at launch.*

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html>

Cloud-init. (s. f.). *Cloud config examples.*

<https://cloudinit.readthedocs.io/en/latest/topics/examples.html>

Cloud-init. (s. f.). *Cloud-init documentation.*

<https://cloudinit.readthedocs.io/en/latest/index.html#>

Github. (s. f.). *plesk/node-express*. <https://github.com/plesk/node-express>

GNU. (s. f.). *Bourne Shell Builtins*.

https://www.gnu.org/software/bash/manual/html_node/Bourne-Shell-Builtins.html

GNU. (s. f.). *GNU «sed»*. <https://www.gnu.org/software/sed/manual/sed.txt>

GNU. (s. f.). *Redirections*.

https://www.gnu.org/software/bash/manual/html_node/Redirections.html

GNU Operating System. (s. f.). *GNU Bash*. <https://www.gnu.org/software/bash>

MongoDB. (s. f.). *Choose which type of deployment is best for you.*

<https://www.mongodb.com/try/download/community>

MongoDB. (s. f.). *Configuration File Options*.

<https://docs.mongodb.com/manual/reference/configuration-options/>

MongoDB. (s. f.). *Install MongoDB Community Edition on Ubuntu.*

<https://docs.mongodb.com/manual/tutorial/install-mongodb-on-ubuntu/>

MongoDB. (s. f.). *Welcome to the MongoDB Documentation.*

<https://docs.mongodb.com>

Nginx. (s. f.). *Products*. <https://docs.nginx.com/>

Página de Express (<https://expressjs.com/>).

Página de NodeJS (<https://nodejs.org/en/>).

Página de NPM (<https://www.npmjs.com/>).

PHP. (s. f.). *Installation on Unix systems.*

<https://www.php.net/manual/en/install.unix.php>

PHP. (s. f.). *PHP Manual.* <https://www.php.net/manual/en/>

Robbins, A. (2010). *Bash Pocket Reference.* O'Reilly Media.

SELinux Project. (s. f.). *Main Page.* https://selinuxproject.org/page/Main_Page

Ubuntu. (s. f.). *Package: mongod (1:3.6.3-0ubuntu1) [universe].*

<https://packages.ubuntu.com/bionic/database/mongodb>

Administración de Sistemas en la Cloud

Administración básica de sistema operativo en Linux

Índice

Esquema	3
Ideas clave	4
5.1. Introducción y objetivos	4
5.2. Sistema de archivos	4
5.3. Usuarios y grupos	12
5.4. Procesos y servicios	18
5.5. Herramientas de sistema	24
5.6. Referencias bibliográficas	37

Esquema

ADMINISTRACIÓN BÁSICA DE SISTEMA OPERATIVO EN LINUX

SISTEMA DE ARCHIVOS

- ▶ El primer nivel de directorios es relativamente estándar y el contenido de cada uno es similar en cualquier distribución Linux.
- ▶ Cualquier cosa es un archivo y está representado en el sistema de archivos: directorios, dispositivos, *sockets*, etc.
- ▶ La asignación de permisos se aplica por usuario, grupo y otros.

USUARIOS Y GRUPOS

- ▶ Los usuarios pertenecen a un grupo principal y cero o varios grupos secundarios.
- ▶ Cada usuario puede tener una carpeta *home*.
- ▶ Cada usuario es asignado a una *shell* para inicio de sesión, aunque puede ser una *shell* especial que no admite *login*.

PROCESOS Y SERVICIOS

- ▶ Cada proceso tiene asignado un PID (controlador proporcional, integral y derivativo).
- ▶ Los servicios son procesos que no terminan inmediatamente y ofrecen una funcionalidad al sistema operativo o al usuario.
- ▶ Kill sirve para enviar señales a los procesos; algunas señales son de obligado cumplimiento.
- ▶ Systemd es un gestor de arranque de procesos que permite administrar servicios.

5.1. Introducción y objetivos

Este tema presentará algunos de los conceptos básicos de la administración de sistemas Linux: sistemas de archivos, usuarios, grupos, etc. Es probable que cualquier instalación de *software* requiera modificaciones y tareas sobre estos objetos. La soltura con la línea de comandos será de ayuda para poder entender los ejemplos de este tema.

Los **objetivos** que se pretenden conseguir son:

- ▶ Conocer los fundamentos de los conceptos básicos de Linux que aparecen de manera recurrente en muchas tareas administrativas.
- ▶ Aprender a manipular los objetos del sistema operativo para automatizar tareas sobre estos.

A continuación, en el vídeo *Administración de usuarios y procesos en Linux*, se verá una demostración de creación, mantenimiento y borrado de usuarios, asignación de permisos.



Accede al vídeo

5.2. Sistema de archivos

Se suele decir que cualquier cosa en Linux es un archivo. Los **archivos** normales no son más que un tipo más de archivo y los **directorios** son archivos que contienen los

nombres de otros archivos. El comando `pwd` ofrece información sobre el directorio actual en la consola. El comando `cd` permite cambiar de directorio.

```
~$ pwd  
/home/ubuntu  
~$ cd /var/log  
/var/log$
```

Árbol de directorios

El directorio raíz se identifica por la barra oblicua (/). Este directorio es la base del árbol de directorios. Al contrario que Windows, todo el sistema operativo de **Linux depende de un único árbol**. No hay unidades de disco C: o D:, como en Windows. En Linux, todas las particiones, unidades y almacenamiento se alojan en algún punto bajo la raíz /. El almacenamiento y las unidades externas se «montan» en un directorio. Este directorio aparece como una carpeta más del árbol. Por ejemplo, la ruta `/media` suele alojar las unidades del CD (disco duro) y las unidades de USB externas, una vez montadas.

El directorio raíz es el inicio de cualquier ruta, o *path*, absoluta. Es posible cambiar de directorio usando **rutas absolutas** desde cualquier directorio. Por ejemplo, `cd /var/log` funcionará aunque la consola se encuentre en `/home/ubuntu`. Las **rutas relativas**, sin embargo, no parten de la raíz, sino del directorio en el que se encuentra la consola en ese momento. Si el directorio actual es `/var`, tanto `cd /var/log` como `cd log` cambiarán al mismo directorio. El símbolo «..» (dos puntos) denota el directorio padre al actual, mientras que «.» (un punto) denota al directorio actual. Ambos pueden usarse en rutas relativas.

```
~ $ cd /var/log  
/var/log $ cd log  
-bash: cd: log: No such file or directory  
/var/log $ cd ..  
/var $ cd log  
/var/log $ cd ./cups
```

```
/var/log/cups $
```

La mayoría de las distribuciones de Linux siguen un **mismo esquema** de árbol de directorios. Aunque hay diferencias, rutas como /bin, /boot, /etc, /home o /tmp son prácticamente una constante en cualquier Linux. La Tabla 1 lista los directorios habituales de la raíz y su contenido.

Directarios habituales de Linux	
/bin	Comandos y binarios de usuario
/boot	Ficheros del gestor de arranque
/dev	Archivos de dispositivos
/etc	Ficheros de configuración
/home	Carpetas <i>home</i> de usuarios
/lib	Librerías compartidas y módulos del <i>kernel</i>
/media	Punto de montaje habitual de medios externos
/mnt	Otro punto de montaje habitual de medios externos
/opt	Software adicional instalado en el sistema
/proc	Detalles del núcleo y de los procesos, almacenados en modo texto
/root	Carpetas <i>home</i> del usuario root
/run	Datos de aplicaciones en ejecución
/sbin	Binarios de sistema
/srv	Datos de servicios provistos por el equipo
/sys	Sistema de ficheros virtual con información del <i>kernel</i>
/tmp	Ficheros temporales
/usr	Utilidades de usuario
/var	Datos transitorios o variables: <i>logs</i> , colas de correo, trabajos de impresión, etc.

Tabla 1. Directarios habituales de Linux. Fuente: elaboración propia.

Para examinar el contenido de los directorios, se puede usar el ya conocido comando ls. Ejecutando ls -la en la *home* de un usuario, se obtiene la siguiente información (Figura 1).

```
1. ubuntu@ubuntu-VirtualBox: ~
~ $ ls -la
total 112
drwxr-xr-x 16 ubuntu ubuntu 4096 May  2 21:22 .
drwxr-xr-x  3 root   root  4096 Apr 22 12:51 ..
-rw-------  1 ubuntu ubuntu 9619 Apr 30 13:22 .bash_history
-rw-r--r--  1 ubuntu ubuntu 220 Apr 22 12:51 .bash_logout
-rw-r--r--  1 ubuntu ubuntu 3771 Apr 22 12:51 .bashrc
drwx----- 13 ubuntu ubuntu 4096 May  2 20:06 .cache
drwx----- 12 ubuntu ubuntu 4096 Apr 28 18:05 .config
drwx-----  3 root   root  4096 Apr 28 18:08 .dbus
drwx-----  3 ubuntu ubuntu 4096 Apr 23 11:51 .gnupg
-rw-------  1 root   root   50 Apr 30 13:05 .lesshst
drwx-----  3 ubuntu ubuntu 4096 Apr 22 12:55 .local
-rw-r--r--  1 ubuntu ubuntu  807 Apr 22 12:51 .profile
-rw-------  1 ubuntu ubuntu 119 Apr 22 19:19 .python_history
drwx-----  2 ubuntu ubuntu 4096 Apr 22 13:13 .ssh
-rw-r--r--  1 ubuntu ubuntu    0 Apr 22 12:56 .sudo_as_admin_successful
-rw-------  1 ubuntu ubuntu 9328 Apr 30 13:15 .viminfo
drwxr-xr-x  2 ubuntu ubuntu 4096 Apr 22 12:55 Desktop
drwxr-xr-x  2 ubuntu ubuntu 4096 Apr 22 12:55 Documents
drwxr-xr-x  2 ubuntu ubuntu 4096 Apr 22 12:55 Downloads
drwxr-xr-x  2 ubuntu ubuntu 4096 Apr 22 12:55 Music
drwxr-xr-x  2 ubuntu ubuntu 4096 Apr 22 12:55 Pictures
drwxr-xr-x  2 ubuntu ubuntu 4096 Apr 22 12:55 Public
drwxr-xr-x  2 ubuntu ubuntu 4096 Apr 22 12:55 Templates
drwxr-xr-x  2 ubuntu ubuntu 4096 Apr 22 12:55 Videos
lrwxrwxrwx  1 ubuntu ubuntu  11 May  2 21:22 a_file -> /tmp/a_file
-rw-rw-r--  1 ubuntu ubuntu 2151 Apr 30 13:15 install.sh
~ $ []
```

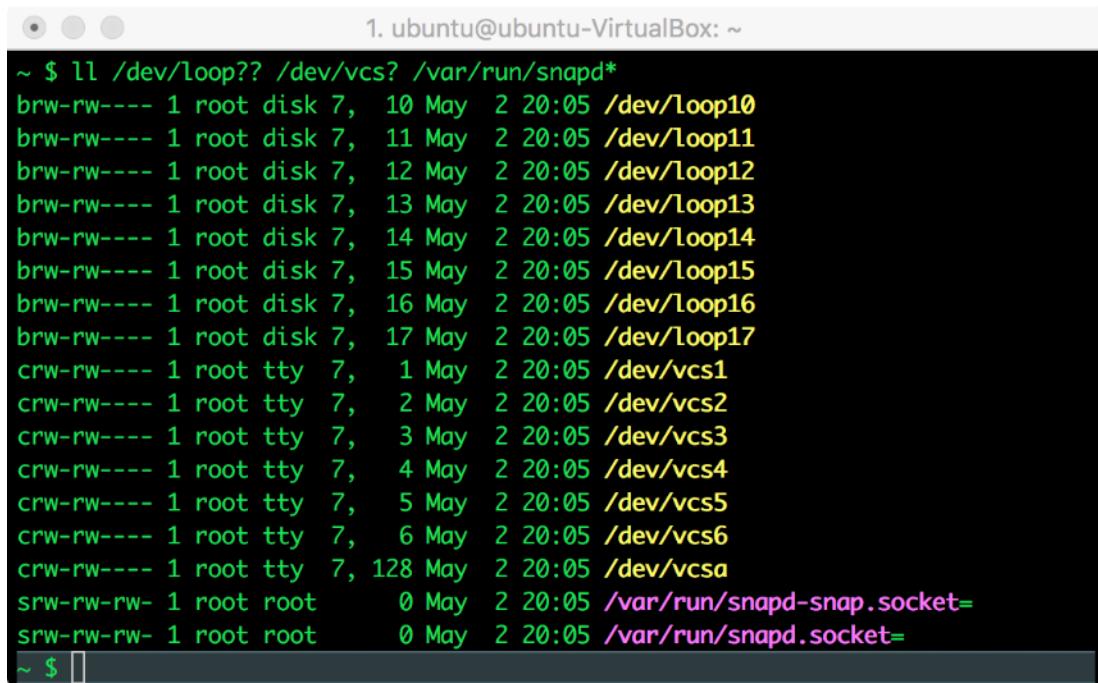
Figura 1. Comando `ls` en la *home* del usuario. Fuente: elaboración propia.

Tipos de archivos

Los ficheros que comienzan con un punto son archivos ocultos. El comando `ls` no lo muestra por defecto, a menos que se invoque con el modificador `-a`. Son archivos normales a todos los efectos. Es habitual que las configuraciones específicas de ciertas aplicaciones se guarden como archivos ocultos en la *home*. Por ejemplo, `.viminfo` son detalles de `vim`, mientras que `.python_history` guarda el histórico de la *shell* de Python.

La primera columna de detalles muestra información sobre el tipo de archivo y los permisos. El primer carácter de la columna indica el tipo de archivo. La mayoría de los archivos de la Figura 1 tiene el *flag* de tipo vacío, con un guion (`-`). Esto significa que son archivos normales. La `d` significa que son directorios, mientras que la `l`

significa que el archivo es un enlace a otro archivo. La c y la b denotan dispositivos de bloque y de carácter, respectivamente, como los mostrados en la Figura 2. Otros tipos de archivos son los *sockets* (s, también en la Figura 2) y las tuberías (p, de *pipe*).



A screenshot of a terminal window titled "1. ubuntu@ubuntu-VirtualBox: ~". The window displays the output of the command "ls -l /dev/loop?? /dev/vcs? /var/run/snapd*". The output lists various device files and sockets. Device files include /dev/loop10 through /dev/loop17, and vcs1 through vcs6. Sockets listed are /var/run/snapd-snap.socket and /var/run/snapd.socket. The terminal prompt is "~ \$".

```
~ $ ls -l /dev/loop?? /dev/vcs? /var/run/snapd*
brw-rw---- 1 root disk 7, 10 May 2 20:05 /dev/loop10
brw-rw---- 1 root disk 7, 11 May 2 20:05 /dev/loop11
brw-rw---- 1 root disk 7, 12 May 2 20:05 /dev/loop12
brw-rw---- 1 root disk 7, 13 May 2 20:05 /dev/loop13
brw-rw---- 1 root disk 7, 14 May 2 20:05 /dev/loop14
brw-rw---- 1 root disk 7, 15 May 2 20:05 /dev/loop15
brw-rw---- 1 root disk 7, 16 May 2 20:05 /dev/loop16
brw-rw---- 1 root disk 7, 17 May 2 20:05 /dev/loop17
crw-rw---- 1 root tty 7, 1 May 2 20:05 /dev/vcs1
crw-rw---- 1 root tty 7, 2 May 2 20:05 /dev/vcs2
crw-rw---- 1 root tty 7, 3 May 2 20:05 /dev/vcs3
crw-rw---- 1 root tty 7, 4 May 2 20:05 /dev/vcs4
crw-rw---- 1 root tty 7, 5 May 2 20:05 /dev/vcs5
crw-rw---- 1 root tty 7, 6 May 2 20:05 /dev/vcs6
crw-rw---- 1 root tty 7, 128 May 2 20:05 /dev/vcsa
srw-rw-rw- 1 root root      0 May 2 20:05 /var/run/snapd-snap.socket=
srw-rw-rw- 1 root root      0 May 2 20:05 /var/run/snapd.socket=
~ $
```

Figura 2. Dispositivos como archivos en /dev y sockets en /var/run. Fuente: elaboración propia.

Sistema de permisos

Los permisos del archivo se indican con los últimos nueve caracteres de la primera columna. En Linux, los permisos se usan para determinar de qué acceso disponen los usuarios y grupos sobre un archivo. El control de permisos sobre archivos y aplicaciones es crítico para la seguridad y el correcto funcionamiento de un *host*. Un permiso erróneo puede suponer un agujero de seguridad (por ejemplo, si todas las subcarpetas de /home permiten lectura al resto de usuarios en un servidor de FTP) o impedir que un servicio funcione correctamente (por ejemplo, si un servidor Nginx se ejecuta con el usuario www-data, pero la carpeta de archivos estáticos solo permite acceso al usuario root). Los tres permisos principales son:

- ▶ Lectura, indicado con el flag r.
- ▶ Escritura/edición, indicado con el flag w.

- ▶ Ejecución, indicado con el flag x.

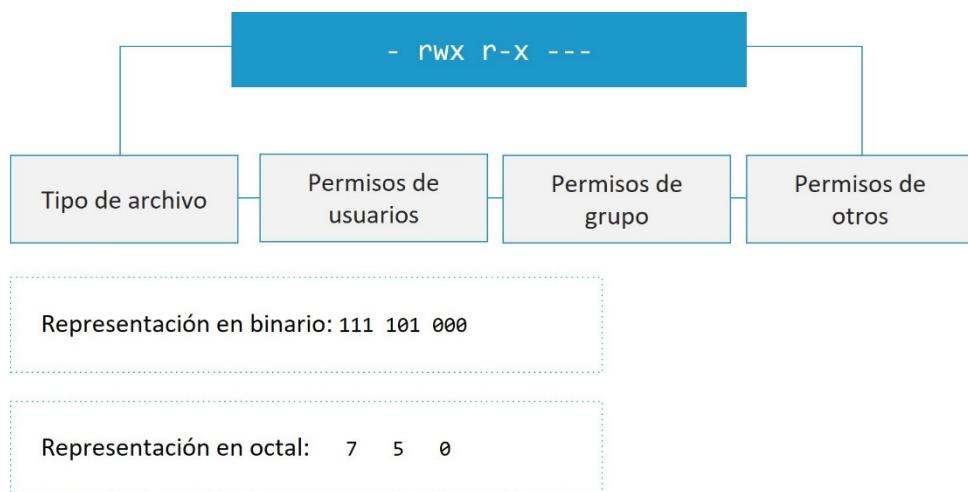


Figura 3. Columna de tipo de archivo y permisos y su representación binaria y octal. Fuente: elaboración propia.

Como su propio nombre indica, el *flag* de lectura permite leer o ver un archivo normal o leer los nombres, pero no los detalles, del contenido de un directorio. El *flag* de escritura permite hacer cambios sobre un archivo o, si se trata de un directorio, permite crear, borrar y renombrar los ficheros del directorio. El *flag* de ejecución permite, efectivamente, la ejecución de un archivo. Los binarios de /bin deben tener el *flag* activo para que se puedan ejecutar. Un *script*, aunque sea un fichero de texto, puede tener el *flag* activo para poder ejecutarlo. Si el *flag* de ejecución se activa en un directorio, es posible «entrar» dentro del directorio usando el comando cd.

Los *flags* están agrupados en **tres clases**: usuario, grupo y otros. Los permisos de usuario aplican al usuario propietario del grupo. Los permisos de grupo describen los permisos que reciben los usuarios miembros del grupo propietario del archivo. El concepto de grupo propietario puede parecer chocante, pero está muy extendido en entornos Linux. La última clase describe los permisos de cualquier otro usuario.

El formato de los *flags* no es arbitrario, ya que se corresponde con la **representación binaria**, de forma que cada *flag* es un *bit*. Si el permiso está activo, el *bit* estará a uno. Una presentación muy habitual de los permisos es agrupar los tres *bits* de cada clase y mostrar la representación octal de esos *bits*. Así, según muestra la Figura 3, si los

tres *flags* están activos, la representación octal de 111 es 7, mientras que, si solo los *flags* de lectura y ejecución están activos, la representación octal de 101 es 5.

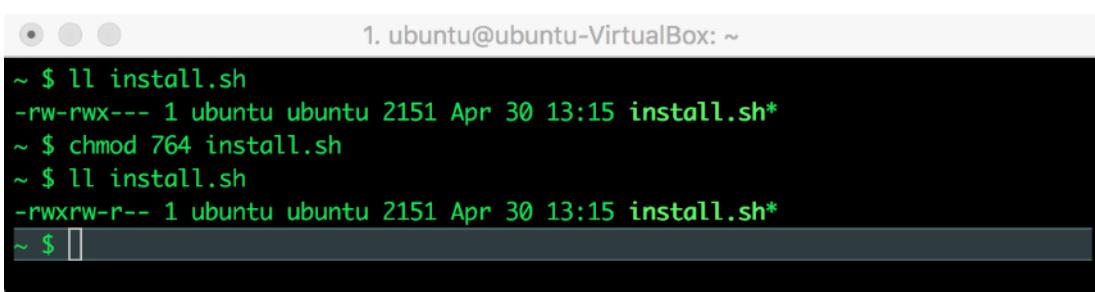
Este formato es muy útil para modificar la configuración de permisos de un archivo. El comando que permite cambiar los permisos es chmod y acepta dos sintaxis:

- ▶ Clase|activar/desactivar|permiso.
- ▶ Representación binaria.

Por ejemplo, dado un archivo con permisos rw-rwx--, es posible activar la ejecución para el usuario propietario, desactivar la ejecución para el grupo y activar la lectura para el resto con los dos comandos siguientes:

```
chmod u+x,g-x,o+r file  
chmod 764 file
```

En el primer comando, cada permiso individual se activa o desactiva para cada clase. En el segundo comando, se usa la representación binaria de los *flags* para definir todos los permisos de una sola vez. La Figura 4 muestra los permisos antes y después de la ejecución de chmod.



A screenshot of a terminal window titled "1. ubuntu@ubuntu-VirtualBox: ~". The terminal shows the following commands and their outputs:
~ \$ ll install.sh
-rw-rwx-- 1 ubuntu ubuntu 2151 Apr 30 13:15 install.sh*
~ \$ chmod 764 install.sh
~ \$ ll install.sh
-rwxrw-r-- 1 ubuntu ubuntu 2151 Apr 30 13:15 install.sh*
~ \$

Figura 4. Cambio de permisos en un archivo. Fuente: elaboración propia.

La Tabla 2 muestra algunos ejemplos de permisos junto a su representación en octal.

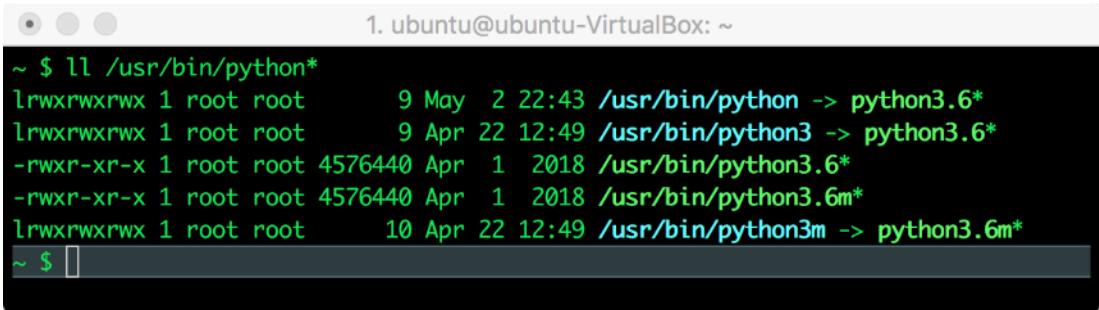
Permisos habituales	
600	r-----
644	rwxr--r--
664	rwxrw-r--
666	rwxrw-rw-
755	rwxr-xr-x
777	Rwxrwxrwx

Tabla 2. Ejemplos de permisos habituales en formato octal. Fuente: elaboración propia.

Enlaces

El fichero de la Figura 1 con el *flag* 1 es un enlace. Hay dos tipos de enlaces: enlaces fuertes y enlaces débiles o simbólicos. Los **enlaces fuertes**, o *hard links*, son referencias reales al archivo. Si se borran todos los enlaces fuertes a un archivo, el archivo se borra también. Un enlace fuerte solo se puede crear en la misma partición que el archivo al que referencia.

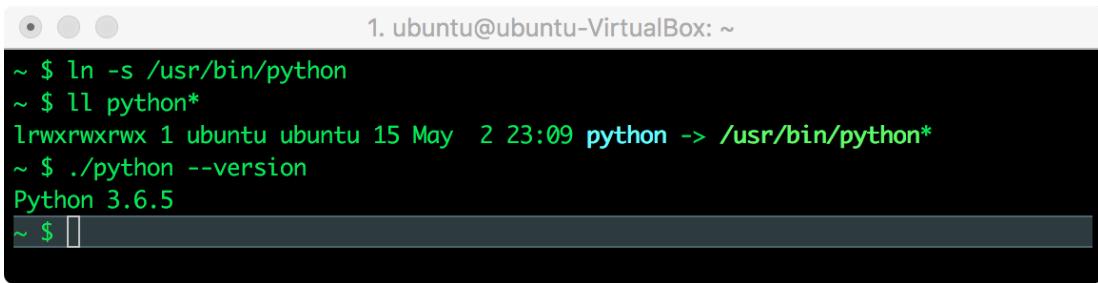
Los **enlaces simbólicos** se pueden borrar sin afectar al fichero al que referencian. Es habitual que los archivos binarios tengan enlaces simbólicos, para poder acceder a ellos con varios nombres o para poder cambiar la versión de un binario sin cambiar el nombre con el que se accede. Por ejemplo, la Figura 5 muestra los ejecutables de Python en la carpeta /usr/bin. La versión instalada de Python es la 3.6, tal como se indica en el binario python3.6. Sin embargo, es posible ejecutarlo tanto con python como con python3. Una actualización a Python 3.8 cambiaría estos enlaces simbólicos, para que apunten al nuevo binario, pero manteniendo los nombres python y python3.



```
1. ubuntu@ubuntu-VirtualBox: ~
~ $ ll /usr/bin/python*
lrwxrwxrwx 1 root root      9 May  2 22:43 /usr/bin/python -> python3.6*
lrwxrwxrwx 1 root root      9 Apr 22 12:49 /usr/bin/python3 -> python3.6*
-rwxr-xr-x 1 root root 4576440 Apr  1  2018 /usr/bin/python3.6*
-rwxr-xr-x 1 root root 4576440 Apr  1  2018 /usr/bin/python3.6m*
lrwxrwxrwx 1 root root     10 Apr 22 12:49 /usr/bin/python3m -> python3.6m*
~ $
```

Figura 5. Enlaces simbólicos. Fuente: elaboración propia.

Los enlaces se crean con el comando `ln`. Los enlaces serán fuertes por defecto y simbólicos con el *flag* `-s`. La manera más sencilla de crear un enlace simbólico en la ruta actual es simplemente indicando la ruta del archivo enlazado (ver Figura 6).



```
1. ubuntu@ubuntu-VirtualBox: ~
~ $ ln -s /usr/bin/python
~ $ ll python*
lrwxrwxrwx 1 ubuntu ubuntu 15 May  2 23:09 python -> /usr/bin/python*
~ $ ./python --version
Python 3.6.5
~ $
```

Figura 6. Creación de enlace simbólico. Fuente: elaboración propia.

5.3. Usuarios y grupos

Linux es un sistema operativo **multiusuario** (Matotek et al., 2017). Esto significa que permite el inicio de sesión de múltiples usuarios simultáneamente, ya sea por la línea de comandos o en una sesión de escritorio. También hay usuarios específicos para ciertos componentes del sistema operativo. Por ejemplo, el servidor web Nginx suele ejecutarse bajo el usuario `nginx` o el usuario `www-data`, según cómo se haya instalado el paquete.

Linux también tiene el concepto de **grupo**. Los usuarios pueden pertenecer a uno o más grupos y suelen agregarse a grupos para dar permisos de acceso a ciertos recursos. Por ejemplo, una carpeta compartida ofrecida en un servidor FTP puede dar

acceso a un grupo concreto. Los miembros de ese grupo podrán leer los ficheros, facilitando la labor administrativa.

Por regla general, al crear un usuario, se crea una carpeta *home*, típicamente debajo de la ruta `/home`. Esta carpeta ofrece un lugar en el que los usuarios pueden guardar sus ficheros, además de ser la ubicación por defecto que muchas aplicaciones usan para guardar las configuraciones específicas de cada usuario. Por ejemplo, las claves para las conexiones SSH se guardan en la carpeta `~/.ssh` y el histórico de Bash se guarda en `~/.bash_profile`. Efectivamente, el símbolo «`~`» hace referencia a la carpeta *home* de un usuario, con independencia de la ruta absoluta de la misma. Un usuario Ubuntu que tenga su carpeta en `/home/ubuntu` podría crear un archivo tanto con `touch ~/new_file` como con `touch /home/ubuntu/new_file`.

Sudo

El comando `sudo` permite la ejecución de comandos administrativos a usuarios distintos de `root`. Las ventajas de Sudo son:

- ▶ Incrementa la seguridad.
- ▶ Permite más granularidad en el control de comandos administrativos.
- ▶ Incorpora auditoría de ejecución.
- ▶ Algunas distribuciones deshabilitan el usuario `root`, por lo que Sudo es la única opción.

Los comandos de creación y modificación de usuarios son ejemplos de comandos administrativos que solo el usuario `root` puede ejecutar. Las distribuciones que deshabilitan el usuario `root` habilitan el comando Sudo para todos los comandos para un usuario normal: Ubuntu lo habilita para el usuario que se crea durante la instalación y, en Amazon Linux, es el usuario `ec2-user` el que tiene permisos.

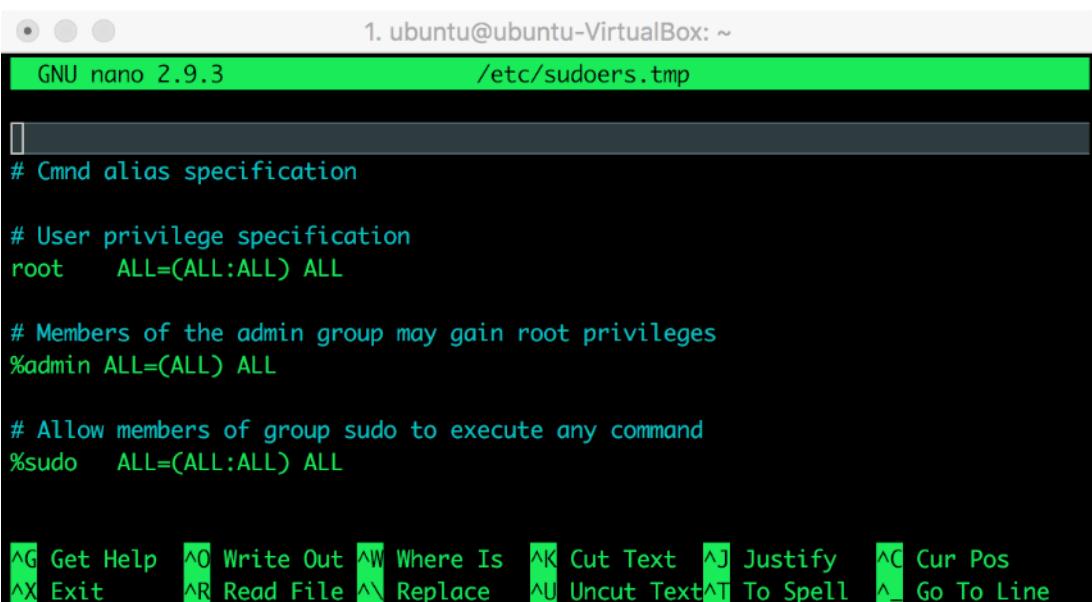
Para modificar permisos de Sudo, hay que editar el fichero `/etc/sudoers` con el comando especial `visudo`. Este comando también necesita privilegios, así que habría

que ejecutarlo con `sudo visudo`. Se abrirá el editor del sistema y se comprobará la sintaxis del fichero. Un fichero `/etc/sudoers` erróneo impedirá que se puedan ejecutar comandos administrativos y, por tanto, ni siquiera se podrá ejecutar `sudo visudo` para arreglar el fichero.

Según el contenido de `/etc/sudoers` de la Figura 7, hay dos formas de dar permisos de Sudo a un usuario:

- ▶ Añadiendo el usuario individualmente con una línea nueva, por ejemplo, `ubuntu ALL=(ALL:ALL) ALL`.
- ▶ Añadir el usuario como miembro de un grupo con permisos, como los grupos `admin` o `sudo`.

La sintaxis del fichero se puede consultar con `man sudoers`.



The screenshot shows a terminal window titled "1. ubuntu@ubuntu-VirtualBox: ~". The title bar says "GNU nano 2.9.3" and the file path is "/etc/sudoers .tmp". The content of the file is as follows:

```
# Cmnd alias specification
# User privilege specification
root    ALL=(ALL:ALL) ALL

# Members of the admin group may gain root privileges
%admin  ALL=(ALL) ALL

# Allow members of group sudo to execute any command
%sudo   ALL=(ALL:ALL) ALL
```

At the bottom of the terminal window, there is a menu bar with various keyboard shortcuts for nano editor commands.

Figura 7. Editor nano tras ejecutar `sudo visudo`. Fuente: elaboración propia.

Administración de usuarios

El comando `useradd` permite crear usuarios nuevos. Durante la creación, se especifican el nombre, la descripción (con el *flag* `-c`), la *shell* por defecto del usuario

(con `-s`) y si se desea crear la *home* del usuario (con `-m`). El usuario estará desactivado y sin contraseña, que habrá que añadir con el comando `passwd`. La Figura 8 muestra la creación de un usuario nuevo, la fijación de la contraseña y el inicio de sesión con este nuevo usuario.

The screenshot shows two terminal windows. The top window is titled '1. ubuntu@ubuntu-VirtualBox: ~' and contains the following command history:

```
~ $ sudo useradd -m -c 'Usuario Nuevo' ubuntu_new
~ $ sudo passwd ubuntu_new
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
~ $
```

The bottom window is titled 'Default (ssh)' and shows the user logging in:

```
~ $ ssh ubuntu_new@192.168.1.131
ubuntu_new@192.168.1.131's password:
Welcome to Ubuntu 18.04.1 LTS (GNU/Linux 4.15.0-29-generic x86_64)

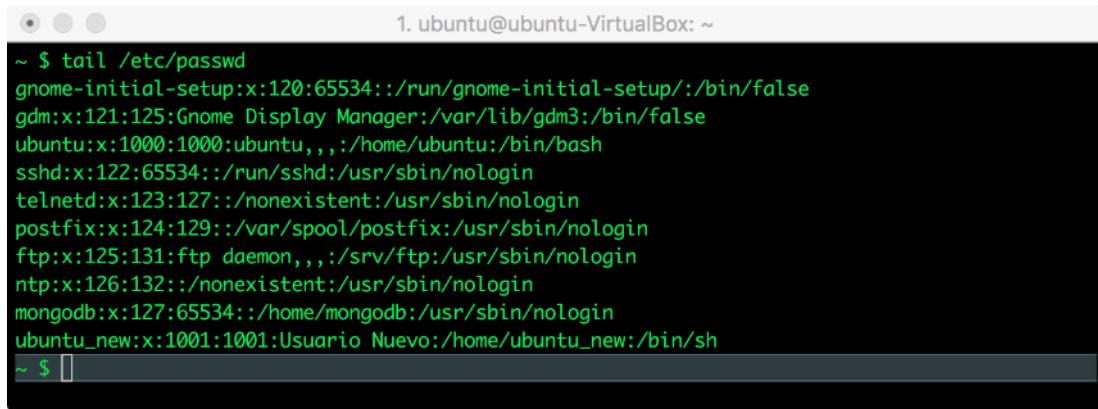
 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

Figura 8. Creación de usuario con `useradd`. Fuente: elaboración propia.

La información del nuevo usuario se almacena en dos ficheros, `/etc/passwd` y `/etc/shadow` (Frampton, s. f.). El primero contiene detalles del usuario: nombre de usuario, ID, ID de grupo, ruta de la *home* y ruta de la *shell* por defecto, entre otros.

Los ID son identificadores numéricos. Las utilidades de administración permiten que los usuarios trabajen con nombres de usuario y de grupo, aunque internamente el sistema trabaje con esos identificadores. La ruta de la *home* puede ser inválida (por ejemplo, `/nonexistent`) si el usuario no dispone de ella. De la misma manera, la ruta de la *shell* puede apuntar a `/usr/sbin/nologin`: esta es una *shell* especial que impide el inicio de sesión, pero registra el intento de inicio de sesión en los *logs*. Un usuario puede no recibir una *shell* en algunos casos. Un servidor FTP, por ejemplo, puede ofrecer acceso a sus usuarios exclusivamente por FTP, pero no por *shell* (se podría

deshabilitar el acceso por SSH completamente, pero esto cierra la puerta al acceso de administradores).

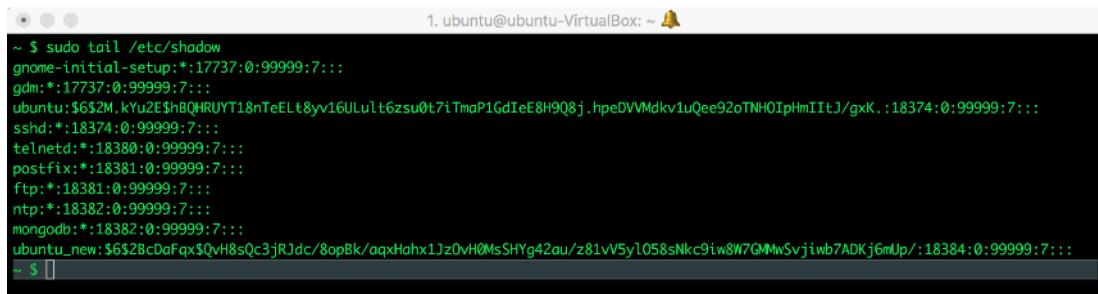


```
1. ubuntu@ubuntu-VirtualBox: ~
~ $ tail /etc/passwd
gnome-initial-setup:x:120:65534::/run/gnome-initial-setup/:/bin/false
gdm:x:121:125:Gnome Display Manager:/var/lib/gdm3:/bin/false
ubuntu:x:1000:1000:ubuntu,,,:/home/ubuntu:/bin/bash
sshd:x:122:65534::/run/sshd:/usr/sbin/nologin
telnetd:x:123:127::/nonexistent:/usr/sbin/nologin
postfix:x:124:129::/var/spool/postfix:/usr/sbin/nologin
ftp:x:125:131:ftp daemon,,,:/srv/ftp:/usr/sbin/nologin
ntp:x:126:132::/nonexistent:/usr/sbin/nologin
mongodb:x:127:65534::/home/mongodb:/usr/sbin/nologin
ubuntu_new:x:1001:1001:Usuario Nuevo:/home/ubuntu_new:/bin/sh
~ $
```

Figura 9. Fichero /etc/passwd. Fuente: elaboración propia.

El fichero /etc/passwd también contiene la contraseña, aunque, en el ejemplo de la Figura 9, todos los usuarios contienen una x en el campo reservado para la misma. Originalmente, las contraseñas se alojaban en ese campo como un *hash* unidireccional, pero este mecanismo era susceptible a ataques de fuerza bruta, especialmente si el fichero era robado, lo que no era imposible, porque el fichero debía ser legible por todos los usuarios.

Para intentar reducir el riesgo, las contraseñas se movieron a un segundo fichero, /etc/shadow, con un *hash* más complejo. Este fichero, además, solo es legible por el usuario root. La Figura 10 muestra el fichero /etc/shadow del mismo sistema que el fichero /etc/passwd de la Figura 9. Solo los usuarios ubuntu y ubuntu_new tienen contraseña.



```
1. ubuntu@ubuntu-VirtualBox: ~
~ $ sudo tail /etc/shadow
gnome-initial-setup:*:17737:0:99999:7:::
gdm:*:17737:0:99999:7:::
ubuntu:$6$2M_kyu2E$8QHRYT18nTeELt8yv16ULult6zsu0t7iTmaP1GdIeE8H9Q8j.hpeDVVMdkv1uQee92oTNHOIpHmIItJ/gxK.:18374:0:99999:7:::
sshd:*:18374:0:99999:7:::
telnetd:*:18380:0:99999:7:::
postfix:*:18381:0:99999:7:::
ftp:*:18381:0:99999:7:::
ntp:*:18382:0:99999:7:::
mongodb:*:18382:0:99999:7:::
ubuntu_new:$6$2BcDfFq$QvH8sQc3jRJdc/8opBk/aqxHohx1Jz0vH0MsSHYg42au/z81vV5y105sNkc9iw8W7QMwSvjlwB7ADKj6nUp/:18384:0:99999:7:::
```

Figura 10. Fichero /etc/shadow. Fuente: elaboración propia.

Administración de grupos

En Linux, cada usuario debe pertenecer al menos a un grupo. La mayoría de las distribuciones crean un nuevo grupo para cada nuevo usuario. Solo este usuario pertenecerá a este grupo, que será el grupo primario del usuario. Los demás grupos de los que sea miembro serán grupos suplementarios. El comando `id` muestra tanto el ID del usuario como los grupos a los que pertenece. En el siguiente ejemplo, el usuario `ubuntu` pertenece a `ubuntu` como grupo principal y a los grupos `cdrom` y `sudo`, entre otros, como suplementarios.

```
$ id ubuntu
uid=1000(ubuntu) gid=1000(ubuntu)
groups=1000(ubuntu),4(adm),24(cdrom),27(sudo),30(dip),46(plugdev),116(lpad
min),126(sambashare)
```

Los grupos suplementarios pueden definirse durante la creación del usuario:

```
$ sudo useradd -m -G sudo,cdrom ubuntu_sudo
$ id ubuntu_sudo
uid=1002(ubuntu_sudo) gid=1002(ubuntu_sudo)
groups=1002(ubuntu_sudo),24(cdrom),27(sudo)
```

También es posible modificar la membresía de grupos una vez creado el usuario con `usermod`.

```
$ sudo usermod -a -G plugdev ubuntu_sudo
$ id ubuntu_sudo
uid=1002(ubuntu_sudo) gid=1002(ubuntu_sudo)
groups=1002(ubuntu_sudo),24(cdrom),27(sudo),46(plugdev)
```

El comando `groupadd` permite crear grupos, mientras que `groupdel` los borra.

```
$ sudo groupadd new_group
$ sudo usermod -a -G new_group ubuntu_sudo
$ id ubuntu_sudo
```

```

uid=1002(ubuntu_sudo) gid=1002(ubuntu_sudo)
groups=1002(ubuntu_sudo),24(cdrom),27(sudo),46(plugdev),1003(new_group)
$ sudo groupdel new_group

```

5.4. Procesos y servicios

Al igual que otros sistemas operativos, en Linux existe el concepto de servicio, también llamado *daemon*. Los **servicios** son procesos que no terminan inmediatamente tras la ejecución, como el comando `pwd`, sino que se ejecutan continuamente, ofreciendo ciertas funcionalidades al sistema operativo o a los usuarios. Algunos ejemplos de servicios son, por ejemplo, el *daemon* de SSH, cuyo proceso se llama `sshd`, o el servidor web Apache, cuyo proceso es `httpd`. El comando `ps aux` muestra todos los procesos en ejecución, tanto servicios como otros procesos.

```

~ $ ps aux | grep -v '^[\.\-\*]'

USER PID %CPU %MEM VSZ RSS TTY STAT START TIME COMMAND
root 1 0.0 0.4 160084 9212 ?
root 228 0.0 1.5 127908 31056 ?
root 238 0.0 0.2 46744 4744 ?
systemd+ 313 0.0 0.3 70744 6344 ?
root 488 0.0 0.4 427256 8848 ?
syslog 492 0.0 0.2 263836 4208 ?
root 502 0.0 0.1 38428 3340 ?
message+ 505 0.0 0.2 51492 5632 ?
stend-activation+ 526 0.0 0.9 580244 18872 ?
root 527 0.0 0.2 44752 5116 ?
root 530 0.0 0.6 518004 13216 ?
root 531 0.0 0.0 4552 752 ?
root 535 0.0 0.8 177638 17308 ?
root 541 0.0 0.2 70688 6036 ?
root 543 0.0 0.4 311004 9092 ?
root 561 0.0 1.2 658696 25736 ?
avahi 564 0.0 0.0 47076 336 ?
root 596 0.0 0.5 311308 11304 ?
root 650 0.0 0.1 28676 2812 ?
root 653 0.0 0.2 72296 5680 ?
root 667 0.0 0.4 308180 8664 ?
whoopsie 729 0.0 0.6 466612 12808 ?
root 730 0.0 0.1 33992 3232 ?
kernoops 751 0.0 0.0 56932 420 ?
kernoops 755 0.0 0.0 56932 416 ?
ubuntu 786 0.0 0.4 77024 8188 ?
ubuntu 795 0.0 0.1 114132 2704 ?
ubuntu 959 0.0 0.3 288380 6688 ?
ubuntu 965 0.0 0.3 212124 6144 tty1
tu gnome-session --session=ubuntu
ubuntu 974 0.0 3.4 409948 70408 tty1

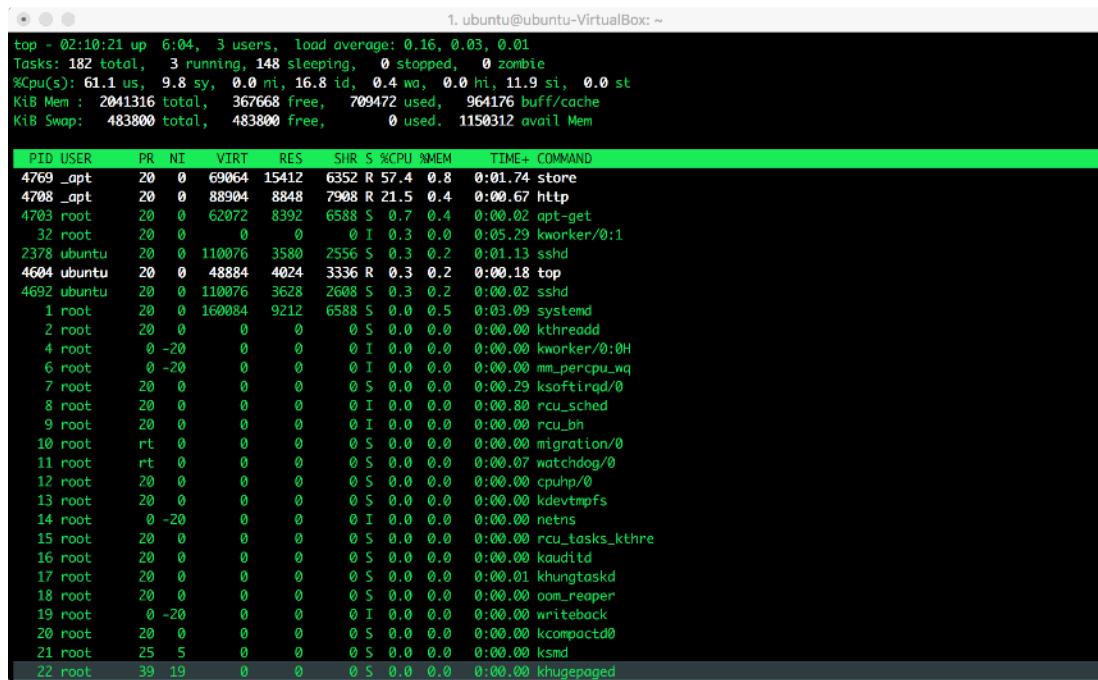
```

Figura 11. Comando `ps aux`. Fuente: elaboración propia.

La lista en la Figura 11 muestra los procesos ordenados por PID, de process ID. El PID se usa para controlar e identificar procesos. El proceso con PID 1, `/sbin/init`, es el primer proceso que arranca y el que se encarga de arrancar otros procesos. El

proceso `init` era originalmente parte del sistema System-V. La mayoría de las distribuciones usan **Systemd** como gestor de arranque de procesos; en esos casos, el proceso con el PID 1 será `/sbin/systemd`. Ubuntu ha conservado el nombre del proceso `init`, aunque utiliza Systemd (Ubuntu usó [Upstart](#) como gestor hasta la versión 16.04).

El comando `top` es una herramienta de monitorización interactiva que muestra los procesos en funcionamiento en el equipo. Al contrario que `ps`, `top` se actualiza cada pocos segundos. Por defecto, `top` ordena los procesos por el uso instantáneo del CPU, por lo que los primeros procesos de la lista son los que más cargan el equipo.



```
1. ubuntu@ubuntu-VirtualBox: ~
top - 02:10:21 up 6:04, 3 users, load average: 0.16, 0.03, 0.01
Tasks: 182 total, 3 running, 148 sleeping, 0 stopped, 0 zombie
%Cpu(s): 61.1 us, 9.8 sy, 0.0 ni, 16.8 id, 0.4 wa, 0.0 hi, 11.9 si, 0.0 st
KiB Mem : 2041316 total, 367668 free, 709472 used, 964176 buff/cache
KiB Swap: 483800 total, 483800 free, 0 used. 1150312 avail Mem

PID USER PR NI VIRT RES SHR %CPU %MEM TIME+ COMMAND
4769 _apt 20 0 69064 15412 6352 R 57.4 0.8 0:01.74 store
4708 _apt 20 0 88904 8848 7908 R 21.5 0.4 0:00.67 http
4703 root 20 0 62072 8392 6588 S 0.7 0.4 0:00.02 apt-get
32 root 20 0 0 0 0 I 0.3 0.0 0:05.29 kworker/0:1
2378 ubuntu 20 0 110076 3580 2556 S 0.3 0.2 0:01.13 sshd
4604 ubuntu 20 0 48884 4024 3336 R 0.3 0.2 0:00.18 top
4692 ubuntu 20 0 110076 3628 2608 S 0.3 0.2 0:00.02 sshd
1 root 20 0 160084 9212 6588 S 0.0 0.5 0:03.09 systemd
2 root 20 0 0 0 S 0.0 0.0 0:00.00 kthreadd
4 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 kworker/0:0H
6 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 mm_percpu_wq
7 root 20 0 0 0 S 0.0 0.0 0:00.29 ksoftirqd/0
8 root 20 0 0 0 0 I 0.0 0.0 0:00.80 rcu_sched
9 root 20 0 0 0 0 I 0.0 0.0 0:00.00 rcu_bh
10 root rt 0 0 0 S 0.0 0.0 0:00.00 migration/0
11 root rt 0 0 0 S 0.0 0.0 0:00.07 watchdog/0
12 root 20 0 0 0 S 0.0 0.0 0:00.00 cpuhp/0
13 root 20 0 0 0 S 0.0 0.0 0:00.00 kdevtmpfs
14 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 netns
15 root 20 0 0 0 S 0.0 0.0 0:00.00 rcu_tasks_kthre
16 root 20 0 0 0 S 0.0 0.0 0:00.00 kaudited
17 root 20 0 0 0 S 0.0 0.0 0:00.01 khungtaskd
18 root 20 0 0 0 S 0.0 0.0 0:00.00 oom_reaper
19 root 0 -20 0 0 0 I 0.0 0.0 0:00.00 writeback
20 root 20 0 0 0 S 0.0 0.0 0:00.00 kcompactd0
21 root 25 5 0 0 0 S 0.0 0.0 0:00.00 ksmd
22 root 39 19 0 0 S 0.0 0.0 0:00.00 khugepaged
```

Figura 12. Comando `top` durante una ejecución de `apt-get update`. Fuente: elaboración propia.

Los procesos pueden terminar por sí mismos si terminan su cometido o si sufren un error que provoque que se cierren con un código de salida diferente de 0. En algunos casos, no obstante, es necesario detener los procesos antes de tiempo. La familia de comandos de `kill` facilitan la tarea de enviar «señales» a los procesos (Van Vugt, 2015). Algunas señales pueden ser ignoradas y, en todo caso, el *software* tiene que estar preparado para atenderlas. Las principales señales se muestran en la Tabla 3.

Señales		
SIGHUP	1	Fuerza un proceso a releer la configuración sin llegar a parar el proceso.
SIGKILL	9	Termina el proceso de manera forzada. El proceso no tiene opción de ignorar la señal y tampoco tiene tiempo para terminar las tareas en curso.
SIGTERM	15	Solicita la terminación del proceso, aunque el proceso puede optar por ignorar esta señal.
SIGUSR1 y SIGUSR2	10 y 12	Envía una señal específica. El proceso tiene que estar diseñado para entender estas señales; y diferentes procesos pueden reaccionar de manera diferente.

Tabla 3. Señales habituales. Fuente: elaboración propia.

El comando `kill` envía una señal a un proceso a partir del PID. Es necesario conocer el PID, que es fácil de obtener a partir de `ps`. Por ejemplo, en la Figura 13, se muestra el PID de un proceso `top` al que se envía la señal `SIGKILL`.

```

2. ubuntu@ubuntu-VirtualBox: ~
ubuntu@ubuntu-VirtualBox:~$ ps aux | grep top
ubuntu 1719 0.0 2.4 770412 48992 tty1 Sl+ May02 0:00 nautilus-desktop
ubuntu 4996 0.3 0.1 48884 3976 pts/1 S+ 02:50 0:00 top
ubuntu 5000 0.0 0.0 21508 1000 pts/3 S+ 02:51 0:00 grep --color=auto top
ubuntu@ubuntu-VirtualBox:~$ kill -9 4996
ubuntu@ubuntu-VirtualBox:~$ ps aux | grep top
ubuntu 1719 0.0 2.4 770412 48992 tty1 Sl+ May02 0:00 nautilus-desktop
ubuntu 5002 0.0 0.0 21508 1000 pts/3 S+ 02:51 0:00 grep --color=auto top
ubuntu@ubuntu-VirtualBox:~$ 

```

Figura 13. Comando `kill` sobre un proceso `top`. Fuente: elaboración propia.

Se puede hacer uso de la sustitución al vuelo de comandos de Bash para obtener el PID del proceso con `pidof` e insertarlo en `kill` en una sola línea:

```
$ sudo kill -9 `pidof top`
```

El comando `killall` es un poco más cómodo de usar para un usuario, ya que permite especificar el nombre del proceso. Como su nombre indica, `killall` terminará todos los procesos con ese nombre. También permite especificar un usuario, por lo que terminará los procesos de ese usuario. Si el usuario ha iniciado una sesión interactiva,

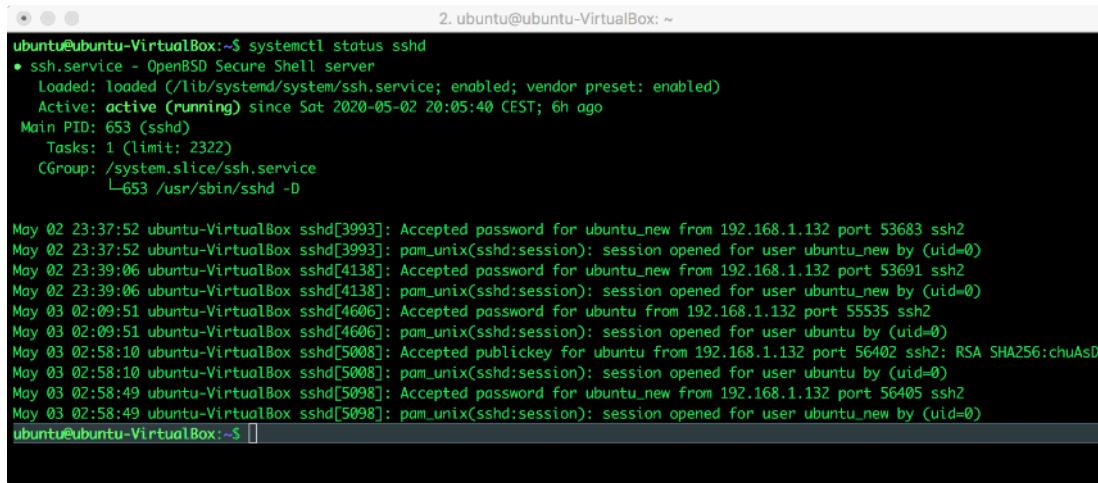
`killall` terminará también el proceso de consola y, por tanto, terminará la sesión del usuario.

```
$ killall -u username  
$ killall -r httpd
```

Finalmente, `pkill` permite terminar procesos con el nombre del proceso. Es también capaz de terminar los procesos hijos de un proceso padre con el *flag* `-P`.

```
$ pkill -P pid
```

Los servicios se pueden administrar como procesos con el comando `kill`, pero es posible usar la funcionalidad de Systemd para obtener más información. La utilidad `systemctl` permite arrancar, parar, recargar y obtener información de los servicios arrancados con Systemd. Por ejemplo, para el servicio de SSH, `systemctl` muestra la información que se encuentra en la Figura 14.



A terminal window titled "2. ubuntu@ubuntu-VirtualBox: ~" displaying the command `systemctl status sshd`. The output shows the service is active and running since May 2, 2020. It lists the main PID (653), tasks (1), and CGroup path (/system.slice/sshd.service). Below this, a log of SSH connections is shown from May 2, 2020, at 02:37:52, including accepted password and publickey logins from 192.168.1.132.

```
ubuntu@ubuntu-VirtualBox:~$ systemctl status sshd  
● ssh.service - OpenBSD Secure Shell server  
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)  
   Active: active (running) since Sat 2020-05-02 20:05:40 CEST; 6h ago  
     Main PID: 653 (sshd)  
       Tasks: 1 (limit: 2322)  
      CGroup: /system.slice/sshd.service  
          └─653 /usr/sbin/sshd -D  
  
May 02 23:37:52 ubuntu-VirtualBox sshd[3993]: Accepted password for ubuntu_new from 192.168.1.132 port 53683 ssh2  
May 02 23:37:52 ubuntu-VirtualBox sshd[3993]: pam_unix(sshd:session): session opened for user ubuntu_new by (uid=0)  
May 02 23:39:06 ubuntu-VirtualBox sshd[4138]: Accepted password for ubuntu_new from 192.168.1.132 port 53691 ssh2  
May 02 23:39:06 ubuntu-VirtualBox sshd[4138]: pam_unix(sshd:session): session opened for user ubuntu_new by (uid=0)  
May 03 02:09:51 ubuntu-VirtualBox sshd[4606]: Accepted password for ubuntu from 192.168.1.132 port 55535 ssh2  
May 03 02:09:51 ubuntu-VirtualBox sshd[4606]: pam_unix(sshd:session): session opened for user ubuntu by (uid=0)  
May 03 02:58:10 ubuntu-VirtualBox sshd[5008]: Accepted publickey for ubuntu from 192.168.1.132 port 56402 ssh2: RSA SHA256:chuAsD  
May 03 02:58:10 ubuntu-VirtualBox sshd[5008]: pam_unix(sshd:session): session opened for user ubuntu by (uid=0)  
May 03 02:58:49 ubuntu-VirtualBox sshd[5098]: Accepted password for ubuntu_new from 192.168.1.132 port 56405 ssh2  
May 03 02:58:49 ubuntu-VirtualBox sshd[5098]: pam_unix(sshd:session): session opened for user ubuntu_new by (uid=0)  
ubuntu@ubuntu-VirtualBox:~$
```

Figura 14. Información del servicio SSH con `systemctl`. Fuente: elaboración propia.

El campo *loaded*, en la segunda línea, indica el fichero utilizado para arrancar el servicio. Este *unit file* no es el binario como tal, sino un fichero de configuración de Systemd que indica, entre otras cosas, condiciones de ejecución, la ruta al binario con los *flags* necesarios para el arranque y ficheros con variables de entorno. El fichero de Systemd de `sshd` se muestra a continuación.

```
$ cat /lib/systemd/system/ssh.service
[Unit]
Description=OpenBSD Secure Shell server
After=network.target auditd.service
ConditionPathExists=!/etc/ssh/sshd_not_to_be_run

[Service]
EnvironmentFile=-/etc/default/ssh
ExecStartPre=/usr/sbin/sshd -t
ExecStart=/usr/sbin/sshd -D $SSHD_OPTS
ExecReload=/usr/sbin/sshd -t
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartPreventExitStatus=255
Type=notify
RuntimeDirectory=sshd
RuntimeDirectoryMode=0755

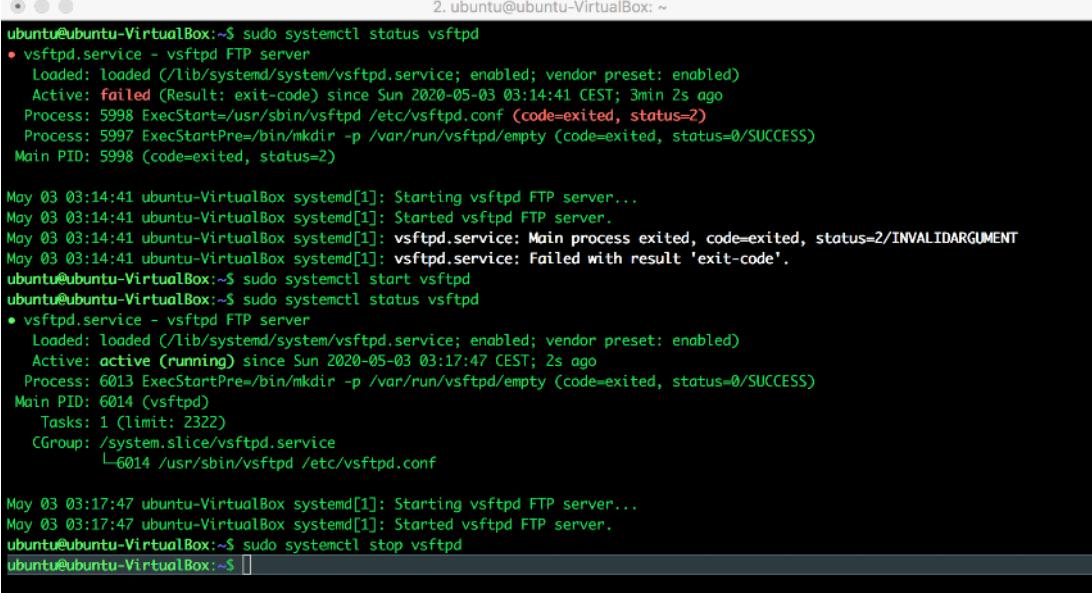
[Install]
WantedBy=multi-user.target
Alias=sshd.service
```

Estos ficheros son editables por un administrador y es posible definir ficheros propios para configurar aplicaciones propias como servicios.

La información provista por `systemctl` va más allá. Indica también el estado del servicio, `active` (`running`) en este caso. El estado puede ser `inactive` (`dead`) si se ha parado el servicio o `failed`, con el código de salida, si el proceso terminó de manera abrupta. Además, las últimas líneas del fichero de `log` se incluyen en la salida de `systemctl status`, lo que puede ser útil si el proceso ha detectado un error antes de detenerse.

Además de ofrecer información, `systemctl` permite arrancar y parar los servicios con `systemctl start` y `systemctl stop`. La Figura 15 muestra ejemplos de ambos comandos. Es necesario identificar el servicio con el nombre y el servicio no siempre

tiene el mismo nombre que el proceso que ejecuta. Es posible listar todos los servicios en ejecución filtrando la salida de `systemctl` con `grep running`, como en la Figura 16, o todos los servicios habilitados (que pueden no estar en ejecución si se han parado manualmente o si han fallado) con `grep enabled`.

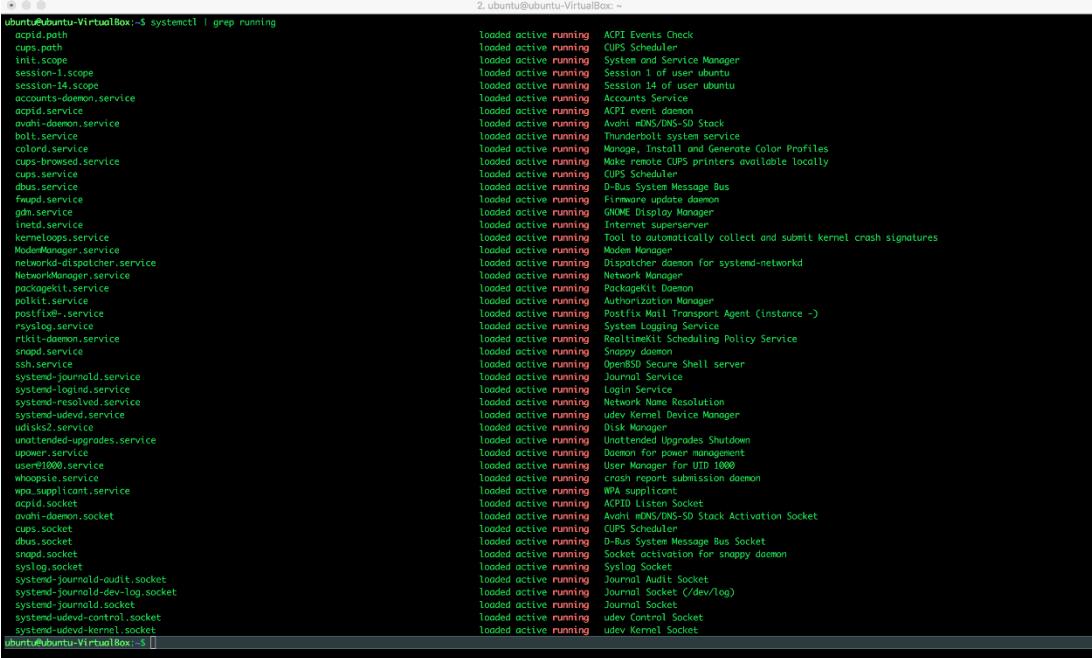


```
ubuntu@ubuntu-VirtualBox:~$ sudo systemctl status vsftpd
● vsftpd.service - vsftpd FTP server
   Loaded: loaded (/lib/systemd/system/vsftpd.service; enabled; vendor preset: enabled)
   Active: failed (Result: exit-code) since Sun 2020-05-03 03:14:41 CEST; 3min 2s ago
     Process: 5998 ExecStart=/usr/sbin/vsftpd /etc/vsftpd.conf (code=exited, status=2)
     Process: 5997 ExecStartPre=/bin/mkdir -p /var/run/vsftpd/empty (code=exited, status=0/SUCCESS)
   Main PID: 5998 (code=exited, status=2)

May 03 03:14:41 ubuntu-VirtualBox systemd[1]: Starting vsftpd FTP server...
May 03 03:14:41 ubuntu-VirtualBox systemd[1]: Started vsftpd FTP server.
May 03 03:14:41 ubuntu-VirtualBox systemd[1]: vsftpd.service: Main process exited, code=exited, status=2/INVALIDARGUMENT
May 03 03:14:41 ubuntu-VirtualBox systemd[1]: vsftpd.service: Failed with result 'exit-code'.
ubuntu@ubuntu-VirtualBox:~$ sudo systemctl start vsftpd
ubuntu@ubuntu-VirtualBox:~$ sudo systemctl status vsftpd
● vsftpd.service - vsftpd FTP server
   Loaded: loaded (/lib/systemd/system/vsftpd.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2020-05-03 03:17:47 CEST; 2s ago
     Process: 6013 ExecStartPre=/bin/mkdir -p /var/run/vsftpd/empty (code=exited, status=0/SUCCESS)
   Main PID: 6014 (vsftpd)
      Tasks: 1 (limit: 2322)
        Group: /system.slice/vsftpd.service
           └─ 6014 /usr/sbin/vsftpd /etc/vsftpd.conf

May 03 03:17:47 ubuntu-VirtualBox systemd[1]: Starting vsftpd FTP server...
May 03 03:17:47 ubuntu-VirtualBox systemd[1]: Started vsftpd FTP server.
ubuntu@ubuntu-VirtualBox:~$ sudo systemctl stop vsftpd
ubuntu@ubuntu-VirtualBox:~$ 
```

Figura 15. Arranque y parada de servicios con `systemctl`. Fuente: elaboración propia.



```
ubuntu@ubuntu-VirtualBox:~$ systemctl | grep running
acpid.path
cups.path
init.scope
session-1.scope
session-14.scope
accounts-dæmon.service
acpid.service
avahi-dæmon.service
bolt.service
colord.service
cups-browsed.service
cups.service
dbus.service
freerdp-serve
gdm.service
inetd.service
kerneloops.service
ModemManager.service
networkd-dispatcher.service
NetworkManager.service
packagekit.service
polkit.service
positiveX-.service
rsyslog.service
runit-.service
snapd.service
ssh.service
systemd-journal.service
systemd-logind.service
systemd-resolved.service
systemd-udevd.service
udisks2.service
unattended-upgrades.service
upower.service
usbmount.service
whoopsie.service
wpa_supplicant.service
acpid.socket
avahi-dæmon.socket
cups.socket
dbus.socket
snapd.socket
syslog.socket
systemd-abstract.socket
systemd-journal-d-dev-log.socket
systemd-journal.socket
systemd-udevd-control.socket
systemd-udevd-kernel.socket
ubuntu@ubuntu-VirtualBox:~$ 
```

Figura 16. Servicios en ejecución con `systemctl`. Fuente: elaboración propia.

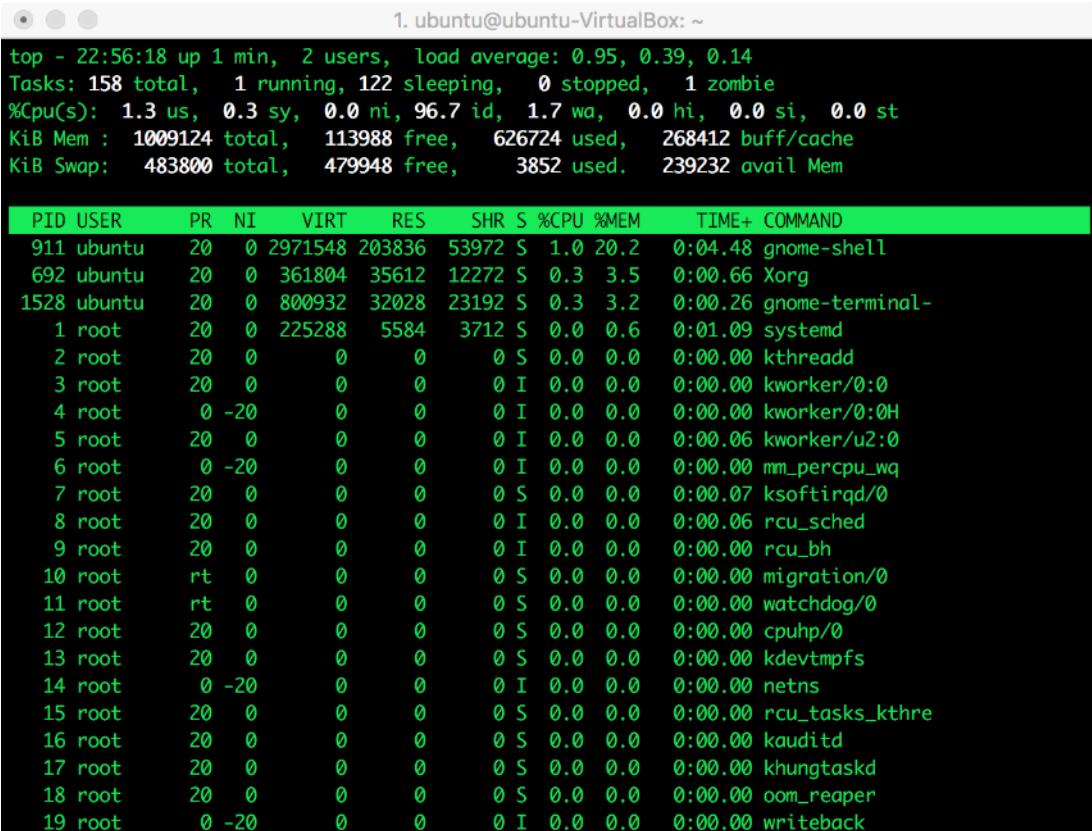
5.5. Herramientas de sistema

Una tarea habitual de los administradores es controlar el rendimiento del sistema.

Algunos de los parámetros habituales son el uso del CPU, memoria, uso de *swap* y espacio libre del disco duro. Las siguientes secciones presentan algunas herramientas de Linux para monitorizar estas métricas.

Top

El comando *top* muestra los comandos con más uso del CPU, ordenados en orden descendente de uso, junto con otras métricas, como uso de memoria, *swap*, etc. La pantalla de *top* se actualiza cada cinco segundos, por defecto, y tendrá un aspecto similar a la Figura 17.



The screenshot shows the terminal window of a desktop environment (Ubuntu) displaying the output of the *top* command. The title bar reads "1. ubuntu@ubuntu-VirtualBox: ~". The output provides system statistics and a detailed list of processes. At the top, it shows:

```
top - 22:56:18 up 1 min, 2 users, load average: 0.95, 0.39, 0.14
Tasks: 158 total, 1 running, 122 sleeping, 0 stopped, 1 zombie
%Cpu(s): 1.3 us, 0.3 sy, 0.0 ni, 96.7 id, 1.7 wa, 0.0 hi, 0.0 si, 0.0 st
KiB Mem : 1009124 total, 113988 free, 626724 used, 268412 buff/cache
KiB Swap: 483800 total, 479948 free, 3852 used. 239232 avail Mem
```

Below this is a table of processes:

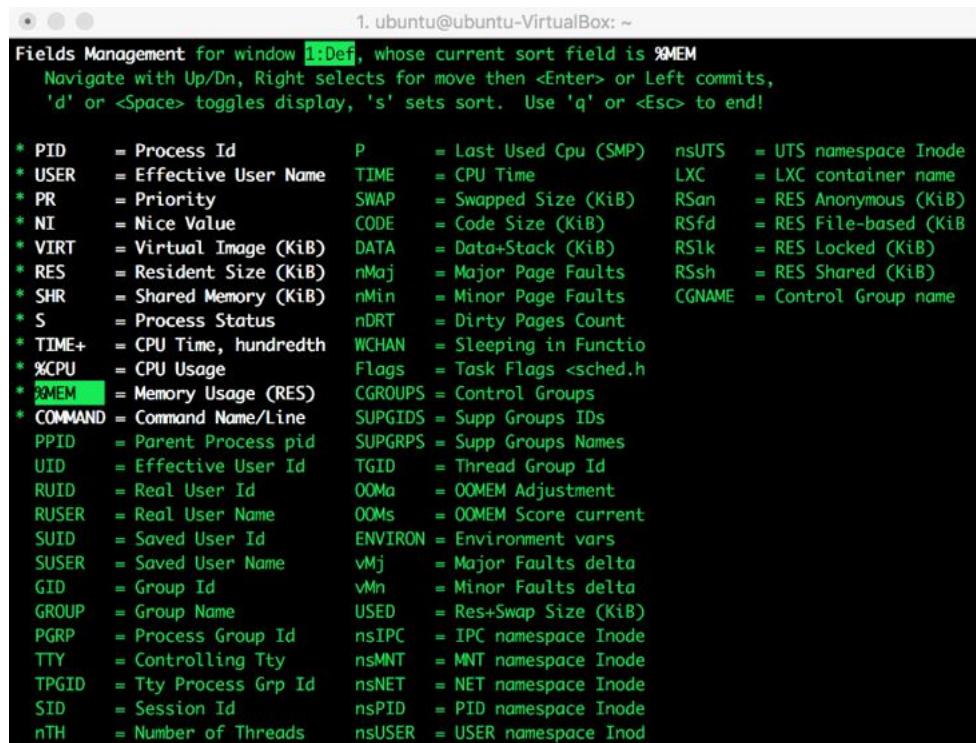
PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
911	ubuntu	20	0	2971548	203836	53972	S	1.0	20.2	0:04.48	gnome-shell
692	ubuntu	20	0	361804	35612	12272	S	0.3	3.5	0:00.66	Xorg
1528	ubuntu	20	0	800932	32028	23192	S	0.3	3.2	0:00.26	gnome-terminal-
1	root	20	0	225288	5584	3712	S	0.0	0.6	0:01.09	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kthreadd
3	root	20	0	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	kworker/0:0H
5	root	20	0	0	0	0	I	0.0	0.0	0:00.06	kworker/u2:0
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
7	root	20	0	0	0	0	S	0.0	0.0	0:00.07	ksoftirqd/0
8	root	20	0	0	0	0	I	0.0	0.0	0:00.06	rcu_sched
9	root	20	0	0	0	0	I	0.0	0.0	0:00.00	rcu_bh
10	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
11	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	watchdog/0
12	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
13	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kdevtmpfs
14	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	netns
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tasks_kthre
16	root	20	0	0	0	0	S	0.0	0.0	0:00.00	kaudittd
17	root	20	0	0	0	0	S	0.0	0.0	0:00.00	khungtaskd
18	root	20	0	0	0	0	S	0.0	0.0	0:00.00	oom_reaper
19	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	writeback

Figura 17. Comando *top* en un sistema de escritorio. Fuente: elaboración propia.

Las primeras líneas muestran un resumen de información del sistema, en este orden:

- ▶ Hora actual, *uptime* del sistema, número de usuarios que han iniciado sesión y la carga media del CPU durante el último minuto, últimos cinco minutos y últimos quince minutos.
- ▶ Número total de procesos y el número en cada estado.
- ▶ Uso del CPU y porcentaje del CPU dedicado a procesos de usuario, a procesos del núcleo del sistema y sin uso.
- ▶ Uso de memoria física: total, libre y en uso, en *bytes*.
- ▶ Uso de memoria virtual: espacio total de *swap*, libre y en uso.

A continuación, la tabla lista los procesos, ordenados por uso de CPU. Es posible cambiar los campos que se muestran, el orden y los campos de resumen. Por ejemplo, para ordenar los procesos por uso de memoria, habría que pulsar f para entrar en el menú de la Figura 18, seleccionar %MEM, pulsar s para ordenar por ese campo y pulsar q para volver a la pantalla de top. También se podrían añadir nuevas columnas, seleccionándolas y pulsando d.



The screenshot shows a terminal window titled "Fields Management for window 1:Def". It displays a list of fields and their meanings. The current sort field is %MEM. The menu includes instructions for navigating (Up/Dn, Right), committing changes (<Enter> or Left), and exiting (d or Space). Fields listed include PID, USER, PR, NI, VIRT, RES, SHR, S, TIME+, %CPU, %MEM, COMMAND, PPID, UID, RUID, RUSER, SUID, SUER, GID, GROUP, PGRP, TTY, TPGID, SID, nTH, P, TIME, SWAP, CODE, DATA, rMaj, rMin, nDRT, WCHAN, Flags, CGROUPS, SUPGIDS, SUPGRPS, TGID, OOMa, OOMs, ENVIRON, vMj, vMn, USED, nsIPC, nsMNT, nsNET, nsPID, nsUSER, nsUTS, LXC, RSan, RSfd, RSlk, RSsh, CGNAME, and UTS namespace Inode, LXC container name, RES Anonymous (KiB), RES File-based (KiB), RES Locked (KiB), RES Shared (KiB), and Control Group name.

Figura 18. Configuración de columnas de top. Fuente: elaboración propia.

La Tabla 4 lista los campos habituales de top y su descripción.

Columnas de top	
PID	ID del proceso.
USER	Nombre del usuario bajo el que se ejecuta el proceso.
PR	Prioridad del proceso.
NI	Valor <i>nice</i> del proceso. Este valor representa la prioridad relativa del proceso: un proceso con un valor alto es <i>nice</i> respecto a otros procesos, porque les cede ciclos de CPU, mientras que un proceso con un valor bajo consume más CPU que el resto.
VIRT	Memoria virtual total en KB.
RES	Memoria física total en KB.
SHR	Memoria compartida.
S	Estado del proceso: dormido (S), hibernado (D), en ejecución (R), zombi (z) o parado (T).
%CPU	Porcentaje de CPU.
%MEM	Porcentaje de memoria física.
TIME+	Tiempo del CPU desde el arranque del proceso.
COMMAND	Comando de arranque.

Tabla 4. Columnas habituales de top. Fuente: elaboración propia.

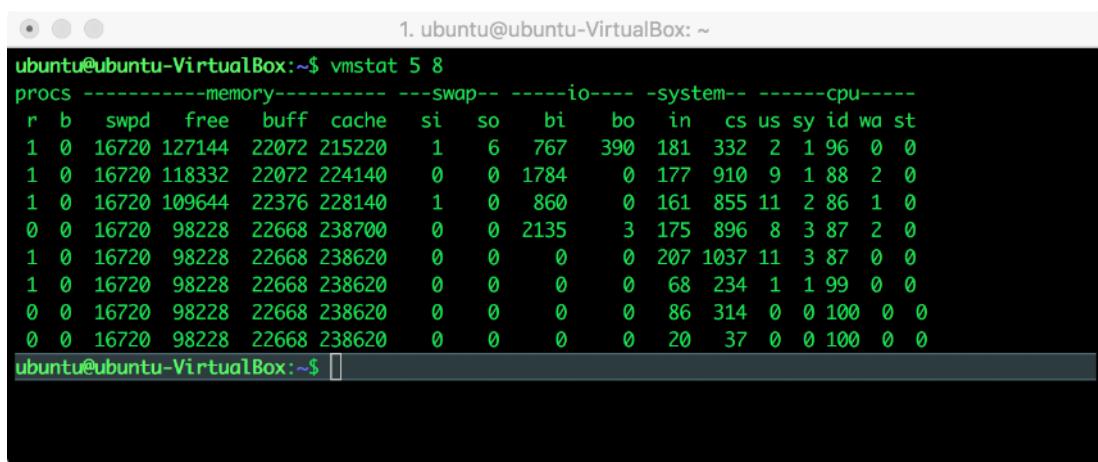
Uptime

El comando `uptime` muestra por consola un resumen del estado del sistema: la hora actual, el tiempo que el sistema ha estado arrancado, el número de usuarios conectados y el uso medio del CPU en el último minuto, últimos cinco minutos y últimos quince minutos.

```
$ uptime  
23:32:56 up 37 min,  2 users,  load average: 0.00, 0.00, 0.05
```

Vmstat

El comando `vmstat` ofrece un resumen de métricas de rendimiento en intervalos definidos por el usuario. En vez de actualizar los valores en la consola, como `top`, imprime los valores de las métricas en una línea periódicamente. El ejemplo de la Figura 19 imprime las métricas cada cinco segundos, hasta un máximo de ocho. Las columnas están organizadas en categorías. La descripción de cada columna se detalla en la Tabla 5.



```
1. ubuntu@ubuntu-VirtualBox: ~
ubuntu@ubuntu-VirtualBox:~$ vmstat 5 8
procs --memory-- swap-- io-- system-- cpu--
 r b swpd free buff cache si so bi bo in cs us sy id wa st
 1 0 16720 127144 22072 215220 1 6 767 390 181 332 2 1 96 0 0
 1 0 16720 118332 22072 224140 0 0 1784 0 177 910 9 1 88 2 0
 1 0 16720 109644 22376 228140 1 0 860 0 161 855 11 2 86 1 0
 0 0 16720 98228 22668 238700 0 0 2135 3 175 896 8 3 87 2 0
 1 0 16720 98228 22668 238620 0 0 0 0 207 1037 11 3 87 0 0
 1 0 16720 98228 22668 238620 0 0 0 0 68 234 1 1 99 0 0
 0 0 16720 98228 22668 238620 0 0 0 0 86 314 0 0 100 0 0
 0 0 16720 98228 22668 238620 0 0 0 0 20 37 0 0 100 0 0
ubuntu@ubuntu-VirtualBox:~$
```

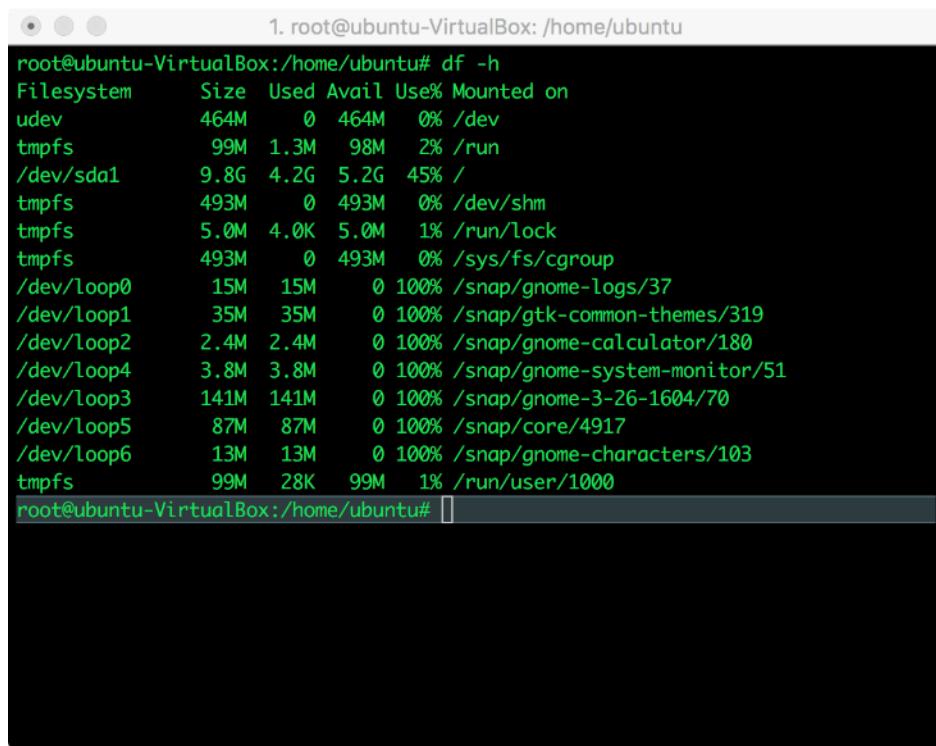
Figura 19. Comando `vmstat`. Fuente: elaboración propia.

Columnas de vmstat	
procs	r: procesos en ejecución b: procesos dormidos
memory	swpd: memoria virtual free: memoria física libre buff: memoria usada como <i>buffers</i> caché: memoria virtual cacheada
swap	si: memoria transferida desde disco (KB) so: memoria transferida a disco (KB)
io	bi: bloques escritos en disco (bloques/s) bo: bloques recibidos de disco (bloques/s)
system	in: interrupciones por segundo cs: cambios de contexto por segundo
cpu	us: uso de CPU de usuario (%) sy: uso de CPU de sistema (%) id: tiempo de CPU sin trabajo (%) wa: tiempo de CPU bloqueada en operaciones de IO (de entrada y salida) (%).

Tabla 5. Columnas de vmstat. Fuente: elaboración propia.

Df

El comando df muestra el espacio disponible en los sistemas de ficheros montados. En la Figura 20 se muestra la salida de df en una instalación de Ubuntu con escritorio. El modificador -h imprime los valores en unidades más legibles para un usuario, como KB, MB, etc.



```
1. root@ubuntu-VirtualBox:/home/ubuntu
root@ubuntu-VirtualBox:/home/ubuntu# df -h
Filesystem      Size  Used Avail Use% Mounted on
udev            464M    0  464M   0% /dev
tmpfs           99M  1.3M  98M   2% /run
/dev/sda1        9.8G  4.2G  5.2G  45% /
tmpfs           493M    0  493M   0% /dev/shm
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           493M    0  493M   0% /sys/fs/cgroup
/dev/loop0       15M   15M    0 100% /snap/gnome-logs/37
/dev/loop1       35M   35M    0 100% /snap/gtk-common-themes/319
/dev/loop2       2.4M   2.4M    0 100% /snap/gnome-calculator/180
/dev/loop4       3.8M   3.8M    0 100% /snap/gnome-system-monitor/51
/dev/loop3       141M  141M    0 100% /snap/gnome-3-26-1604/70
/dev/loop5       87M   87M    0 100% /snap/core/4917
/dev/loop6       13M   13M    0 100% /snap/gnome-characters/103
tmpfs           99M   28K  99M   1% /run/user/1000
root@ubuntu-VirtualBox:/home/ubuntu#
```

Figura 20. Comando df -h. Fuente. Elaboración propia.

Directorio /proc

En la sección del sistema de ficheros, se mencionó que el directorio /proc es un directorio especial, ya que contiene archivos con información del sistema operativo. De hecho, no solo se puede obtener información, sino que es posible cambiar los parámetros del núcleo de Linux editando los archivos contenidos de /proc, modificando así el comportamiento del sistema.

El sistema de archivos /proc no es un directorio real en el disco duro, sino una colección de estructuras de datos en la memoria, administrada por el núcleo, que aparece como un conjunto de directorios y archivos. El propósito de /proc (también llamado **sistema de archivos de procesos**) es ofrecer acceso a la información sobre el sistema operativo y sobre todos los procesos que se encuentran en ejecución.

Los archivos de /proc son accesibles, como cualquier otro, pero es necesario conocer el significado y la sintaxis de ellos para interpretar la información. Por ejemplo, los

comandos cat o less mostrarán el contenido de los archivos y ls muestra una lista con los archivos disponibles.

```
root@ubuntu-VirtualBox:/proc# ls /proc
1 1051 1056 114 1289 16 2 25 30 410 669 79 921  buddyinfo  fs  locks  slabinfo  vmstat
1003 1062 1140 13 165 20 258 32 413 684 88 946  cgroups  iomem  meminfo  softirqs  zoneinfo
1011 1066 1141 1373 166 2056 26 34 434 7 81 947  cmdline  ioports  misc  swaps
1016 1067 1143 1374 167 2095 260 35 435 753 82 959  consoles  irq  modules  sys
1020 1077 1150 14 1695 21 262 376 438 758 84 963  cpuinfo  kallsyms  mounts  sysrq-trigger
1025 1081 1165 1408 17 2161 263 378 469 759 88 965  crypto  kcore  mtrr  sysvipc
1029 1084 1182 1409 1784 22 264 383 494 77 880 969  devices  key-users  net  thread-self
1033 1085 1187 1410 18 220 265 399 586 772 883 97  diskstats  keys  pagetypeinfo  timer_list
1037 1087 1194 1421 1827 2232 266 4 507 776 888 979  dma  kmsg  partitions  tty
1040 1089 12 15 188 2241 267 400 593 778 897 983  driver  kpagedgroup  sched_debug  uptime
1044 1091 1210 1508 1886 23 268 402 6 78 9 992  execdomains  kpagedcount  schedstat  version
1046 1095 1217 152 189 24 27 405 605 782 902  acpi  fb  kpagedflags  scsi  version_signature
1047 11 1229 1526 19 243 28 409 627 786 905  asound  filesystems  loadavg  self  vmallocinfo
root@ubuntu-VirtualBox:/proc#
```

Figura 21. Listado de archivos de /proc. Fuente: elaboración propia.

El primer conjunto de directorios (indicado por el *flag d* en la salida de ls -l /proc o en diferente color en la Figura 21) representa los procesos que se ejecutan actualmente en su sistema. Cada directorio que corresponde a un proceso tiene el PID como nombre. Los archivos de cada directorio contienen información sobre el proceso: el binario, la línea de comandos con la que se ha ejecutado, las variables de entorno, uso de memoria, etc. La Figura 22 muestra algunos de estos ficheros para el proceso de un servidor FTP.

```
root@ubuntu-VirtualBox:/proc/662# ll exe
lrwxrwxrwx 1 root root 0 May 10 07:43 exe -> /usr/sbin/vsftpd*
root@ubuntu-VirtualBox:/proc/662# cat environ ; echo
LANG=en_US.UTF-8LC_ADDRESS=es_ES.UTF-8LC_IDENTIFICATION=es_ES.UTF-8LC_MEASUREMENT=es_ES.UTF-8LC_MONETARY=es_ES.UTF-8LC_NAME=es_ES.UTF-8LC_NUMERIC=es_ES.UTF-8LC_PAPER=es_ES.UTF-8LC_TELEPHONE=es_ES.UTF-8LC_TIME=es_ES.UTF-8PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/binINVOCATION_ID=14a737c22ab54f5eb1cc0f87a19a6905JOURNAL_STREAM=9:19683
root@ubuntu-VirtualBox:/proc/662# cat cmdline ; echo
/usr/sbin/vsftpd /etc/vsftpd.conf
root@ubuntu-VirtualBox:/proc/662#
```

Figura 22. Detalles del proceso del servidor FTP. Fuente: elaboración propia.

Otro archivo interesante es /proc/cpuinfo. Este contiene las características de los procesadores del equipo: fabricante, modelo, conjuntos de instrucciones, frecuencia, etc.

```

# cat /proc/cpuinfo
processor          : 0
vendor_id         : GenuineIntel
cpu family        : 6
model             : 158
model name        : Intel(R) Core(TM) i7-7820HQ CPU @ 2.90GHz
stepping          : 9
cpu MHz           : 2903.998
cache size        : 8192 KB
physical id       : 0
siblings          : 1
core id           : 0
cpu cores         : 1
apicid            : 0
initial apicid   : 0
fpu               : yes
fpu_exception     : yes
cpuid level       : 22
wp                : yes
flags              : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
                      mca cmov pat pse36 clflush mmx fxsr sse sse2 syscall nx rdtscp lm constant_tsc
                      rep_good nopl xtopology nonstop_tsc cpuid pnpi pclmulqdq monitor ssse3 cx16
                      sse4_1 sse4_2 x2apic movbe popcnt aes xsave avx rdrand hypervisor lahf_lm
                      abm 3dnowprefetch ptib avx2 rdseed clflushopt
bugs              : cpu_meltdown spectre_v1 spectre_v2 spec_store_bypass
bogomips          : 5807.99
clflush size      : 64
cache_alignment    : 64
address sizes     : 39 bits physical, 48 bits virtual
power management:

```

La Tabla 6 describe algunos de los archivos y directorios de /proc.

Contenido de /proc	
/proc/acpi	DIRECTORIO con información sobre ACPI, la interfaz de control de energía.
/proc/cmdline	Línea de comandos usada para el arranque del <i>kernel</i> .
/proc/devices	Dispositivos de bloque y de carácter del sistema.
/proc/filesystems	Lista de sistemas de archivos soportados.
/proc/kcore	Imagen de la memoria física.
/proc/locks	Bloqueos (<i>locks</i>) del sistema operativo en curso.
/proc/meminfo	Información sobre la memoria física y virtual.
/proc/mounts	Lista de sistemas de archivos montados.
/proc/net	DIRECTORIO con información sobre las interfaces y los protocolos de red.
/proc/partitions	Lista de particiones.
/proc/scsi/scsi	Información de los dispositivos SCSI.
/proc/sys	DIRECTORIO con información del sistema. El contenido se puede editar para configurar parámetros específicos del sistema operativo.
/proc/uptime	Tiempo de ejecución del sistema.
/procversión	Versión del núcleo de Linux.

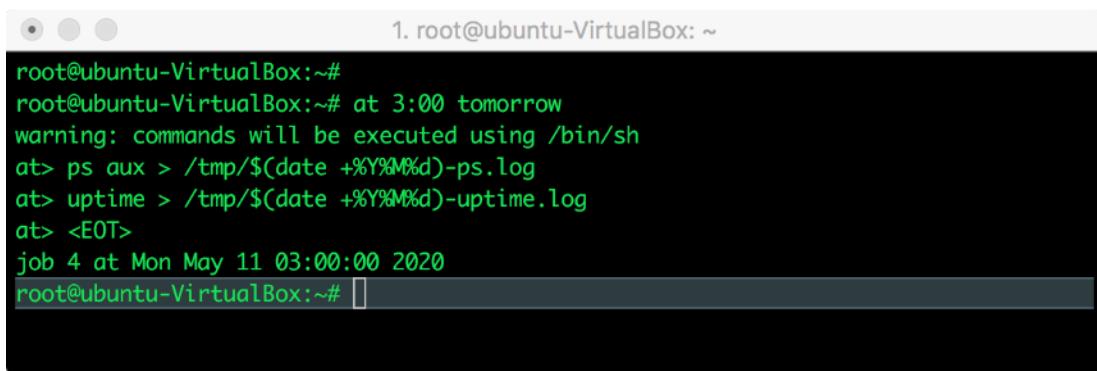
Tabla 6. Algunos ficheros y directorios de /proc. Fuente: elaboración propia.

Programación de tareas

Ciertos programas de Linux no están pensados para ejecutarse continuamente como un servicio, pero es necesario ejecutarlos en un instante futuro dado o periódicamente. Algunas de estas tareas pueden ser copia de seguridad, análisis de *logs* o descarga de grandes ficheros (por ejemplo, ficheros de actualización o sincronización de *mirrors*) en horario nocturno.

El comando **at** programa comandos para su ejecución en un momento dado. El servicio atd se encarga de ejecutar los comandos programados con at. Estos trabajos se ejecutarán en el momento indicado, tras lo cual at los borrará de la lista. Los pasos para programar un trabajo son los siguientes:

- ▶ Ejecutar at <tiempo>. El formato para indicar el instante es muy versátil y acepta formatos como:
 - at 21:30.
 - at now.
 - at now + 15 minutes.
 - at now + 4 hours.
 - at noon.
 - at now next hour (dentro de 60 minutos).
 - at now next day (a la misma hora del día siguiente).
 - at 17:00 tomorrow.
 - at 3:00 Jan 18, 2021.
- ▶ Se abrirá una *shell* de at en la que se introducen los comandos, uno por línea.
- ▶ Para cerrar la consola de at, hay que pulsar Ctrl+D, una vez escritos los comandos.



The screenshot shows a terminal window with the title '1. root@ubuntu-VirtualBox: ~'. The terminal output is as follows:

```
root@ubuntu-VirtualBox:~#
root@ubuntu-VirtualBox:~# at 3:00 tomorrow
warning: commands will be executed using /bin/sh
at> ps aux > /tmp/$(date +%Y%m%d)-ps.log
at> uptime > /tmp/$(date +%Y%m%d)-uptime.log
at> <EOT>
job 4 at Mon May 11 03:00:00 2020
root@ubuntu-VirtualBox:~#
```

Figura 23. Programación de comandos con at. Fuente: elaboración propia.

La Figura 23 muestra cómo se añaden dos comandos para su ejecución a las 3:00 a. m. del día siguiente. La lista de trabajos programados se puede consultar con **atq** y se pueden borrar con **atrm <id>**.

```
$ atq
3      Mon May 11 03:00:00 2020 a root
2      Sun May 10 21:31:00 2020 a root
```

Este comando no permite, sin embargo, programar tareas que se ejecuten periódicamente de manera automática. Para ello, se puede usar el servicio **cron** y el

comando **crontab**. *Cron* no permite trabajos con más de un comando, pero siempre se puede escribir un *script* con los comandos necesarios. El servicio comprueba los trabajos disponibles cada minuto y ejecuta los que deban ejecutarse en ese momento.

Los pasos para programar un trabajo de cron son:

- ▶ Preparar el *script* o el comando que se ejecutará.
- ▶ Preparar un archivo de texto con la programación y el comando. El archivo puede contener más de una tarea, cada una con su programación.
- ▶ Ejecutar **crontab <fichero>** para confirmar la definición.
- ▶ Confirmar que el trabajo está definido con **crontab -l**.

El formato de definición de tareas es el siguiente:

```
5 0 * * * ps aux > /tmp/$(date +%Y%M%d$h%m)-ps.log
```

Las cinco primeras columnas de la línea definen la programación: cada campo representa una unidad temporal y el valor indica en qué momento se «activa» ese campo. Cada campo puede contener un número, una lista de números separadas por comas, una pareja de números con un guion representando un intervalo o un asterisco, que indica todos los valores posibles. La Tabla 7 lista la unidad y los posibles valores de cada campo.

Campos de programación de cron		
1	Minutos	0-59
2	Hora	0-23
3	Día del mes	0-31
4	Mes	1-12 o las primeras letras del mes: <i>Jan, Feb</i> , etc.
5	Día de la semana	0-6, de domingo a sábado, o las primeras letras: <i>Sun, Mon, Tue</i> , etc.

Tabla 7. Campos de la programación de cron. Fuente: elaboración propia.

Este sistema puede parecer complicado, así que la Tabla 8 tiene ejemplos de cómo interpretar la sintaxis.

Ejemplos de programación de cron	
5 0 * * *	Todos los días a las 00:05
0,15,30,45 * * * *	Cada 15 minutos
0 3 1 * *	Mensualmente, el día 1 de cada mes, a las 3:00
* * * * *	Cada minuto
10 0 * * 6	Los sábados a las 00:10

Tabla 8. Ejemplos de cron. Fuente: elaboración propia.

Cuando un usuario ejecuta `crontab -l`, el comando solo muestra los trabajos programados por él mismo. Por ejemplo, si el usuario root define el trabajo del ejemplo, su listado de trabajo sería el siguiente:

```
$ crontab -l
5 0 * * * ps aux > /tmp/$(date +%Y%M%d$h%m)-ps.log
```

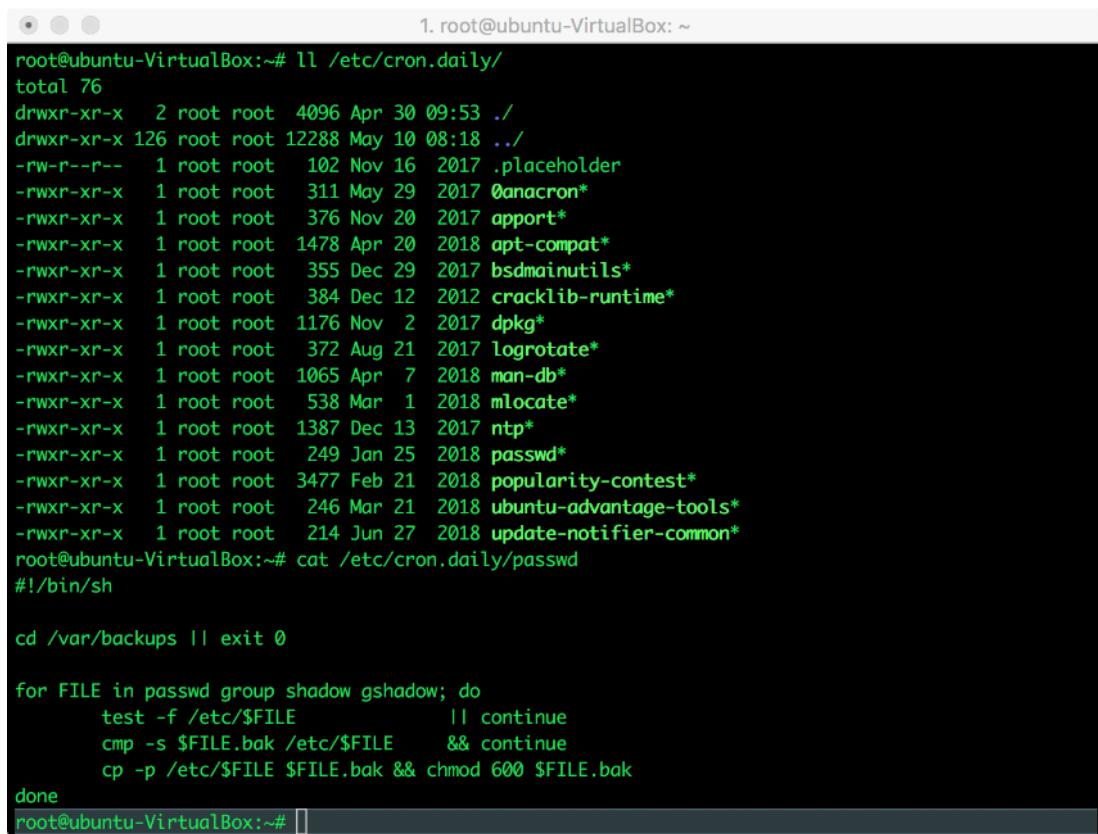
Es decir, exactamente el contenido del fichero con el que se ha definido el trabajo. Sin embargo, el fichero `/etc/crontab` contiene la programación de trabajos de

sistema. Por ejemplo, en un equipo Ubuntu, el fichero /etc/crontab por defecto contiene lo siguiente:

```
$ cat /etc/crontab
SHELL=/bin/sh
PATH=/usr/local/sbin:/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin

# m h dom mon dow user    command
17 * * * * root    cd / && run-parts --report /etc/cron.hourly
25 6 * * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.daily )
47 6 * * 7 root    test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.weekly )
52 6 1 * * root    test -x /usr/sbin/anacron || ( cd / && run-parts --report
/etc/cron.monthly )
```

Además de unas variables de entorno que se aplicarán a cada uno de los trabajos, el fichero contiene cuatro trabajos, que corresponden con tareas horarias, diarias, semanales y mensuales. El comando `run-parts` ejecutará los *scripts* contenidos en cada una de las carpetas `/etc/cron.*`, siguiendo la programación del fichero crontab. Los servicios de sistema pueden aprovechar esta funcionalidad para añadir *scripts* recurrentes en estas carpetas y así delegar la ejecución periódica a cron. Por ejemplo, en la Figura 24, se muestran las tareas diarias. Una de ellas, `/etc/cron.daily/passwd`, crea copias de seguridad de los ficheros de las cuentas de usuario y grupo en la carpeta `/var/backup`.



```

1. root@ubuntu-VirtualBox: ~
root@ubuntu-VirtualBox:~# ll /etc/cron.daily/
total 76
drwxr-xr-x  2 root root  4096 Apr 30 09:53 .
drwxr-xr-x 126 root root 12288 May 10 08:18 ../
-rw-r--r--  1 root root   102 Nov 16 2017 .placeholder
-rwxr-xr-x  1 root root   311 May 29 2017 0anacron*
-rwxr-xr-x  1 root root   376 Nov 20 2017 apport*
-rwxr-xr-x  1 root root  1478 Apr 20 2018 apt-compat*
-rwxr-xr-x  1 root root   355 Dec 29 2017 bsdmainutils*
-rwxr-xr-x  1 root root   384 Dec 12 2012 cracklib-runtime*
-rwxr-xr-x  1 root root  1176 Nov  2 2017 dpkg*
-rwxr-xr-x  1 root root   372 Aug 21 2017 logrotate*
-rwxr-xr-x  1 root root  1065 Apr  7 2018 man-db*
-rwxr-xr-x  1 root root   538 Mar  1 2018 mlocate*
-rwxr-xr-x  1 root root  1387 Dec 13 2017 ntp*
-rwxr-xr-x  1 root root   249 Jan 25 2018 passwd*
-rwxr-xr-x  1 root root  3477 Feb 21 2018 popularity-contest*
-rwxr-xr-x  1 root root   246 Mar 21 2018 ubuntu-advantage-tools*
-rwxr-xr-x  1 root root   214 Jun 27 2018 update-notifier-common*
root@ubuntu-VirtualBox:~# cat /etc/cron.daily/passwd
#!/bin/sh

cd /var/backups || exit 0

for FILE in passwd group shadow gshadow; do
    test -f /etc/$FILE           || continue
    cmp -s $FILE.bak /etc/$FILE   && continue
    cp -p /etc/$FILE $FILE.bak && chmod 600 $FILE.bak
done
root@ubuntu-VirtualBox:~# []

```

Figura 24. Tareas diarias en /etc/cron.daily. Fuente: elaboración propia.

5.6. Referencias bibliográficas

Frampton, S. (s. f.). 6.6. Linux Password & Shadow File Formats. En *Linux Administration Made Easy*.

<https://www.tldp.org/LDP/lame/LAME/linux-admin-made-easy/shadow-file-formats.html>

Matotek, D., Turnbull, J. y Lieverdink, P. (2017). *Pro Linux System Administration: Learn to Build Systems for Your Business Using Free and Open Source Software* (2.ª ed.). Apress.

Página de Upstart (<http://upstart.ubuntu.com/>).

Van Vugt, S. (2015). *Beginning the Linux Command Line* (2.ª ed.). Apress.

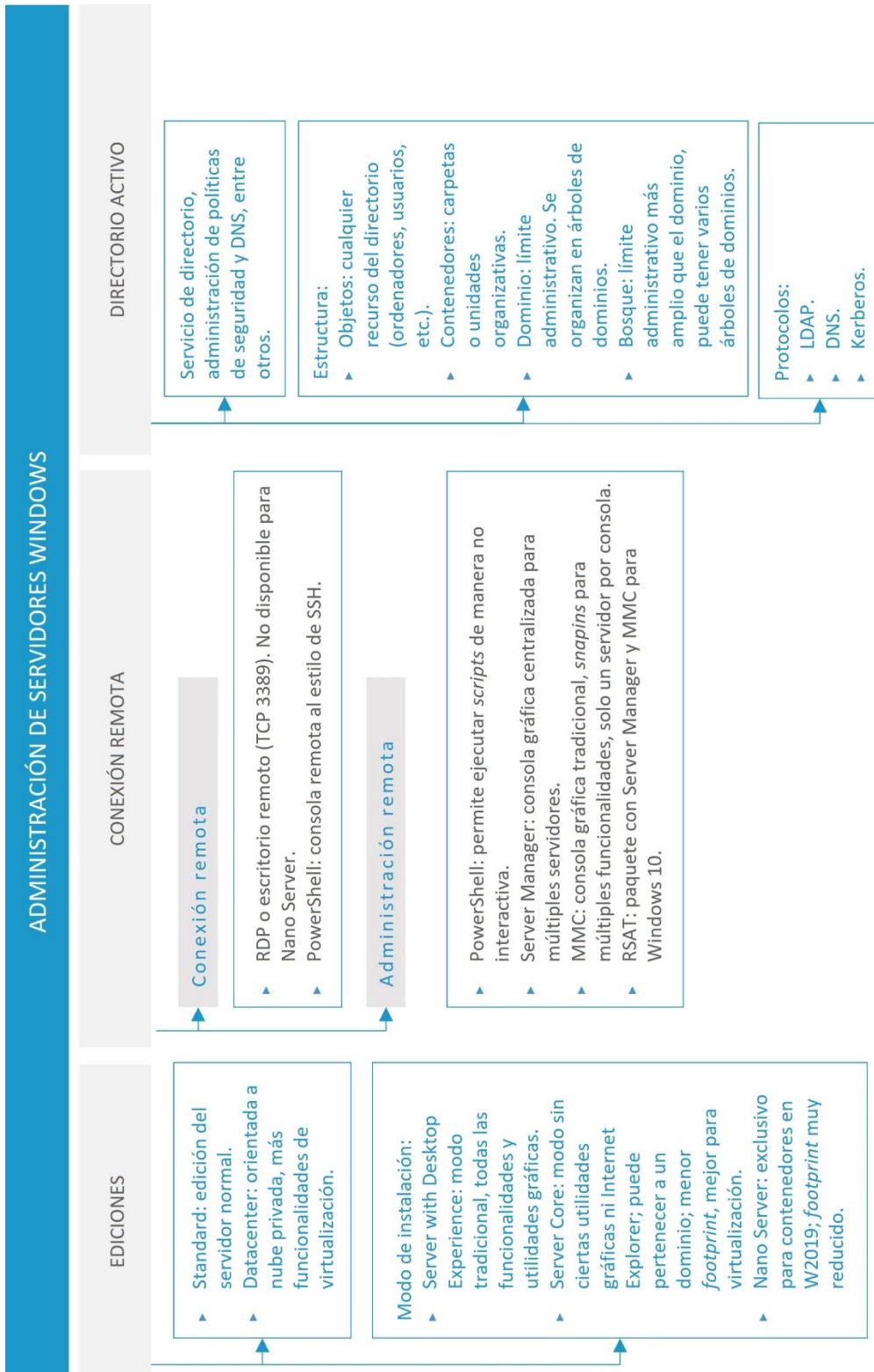
Administración de Sistemas en la Cloud

Administración de servidores Windows

Índice

Esquema	3
Ideas clave	4
6.1. Introducción y objetivos	4
6.2. Ediciones de Windows	4
6.3. Acceso remoto	8
6.4. Directorio Activo	14
6.5. Creación de un dominio	20
6.6. Referencias bibliográficas	27

Esquema



6.1. Introducción y objetivos

Este tema presentará algunos de los conceptos básicos de la administración de sistemas Windows: sistemas de archivos, usuarios, grupos, etc. Es probable que cualquier instalación de *software* requiera de modificaciones y tareas sobre estos objetos.

Los **objetivos** que se pretenden conseguir son:

- ▶ Conocer los fundamentos de los conceptos básicos de Windows que aparecen de manera recurrente en muchas tareas administrativas.
- ▶ Aprender a manipular los objetos del sistema operativo para automatizar tareas sobre los mismos.

A continuación, en el vídeo *Instalación de Windows Server*, se ofrece una guía paso a paso para instalar una máquina con Windows Server y para conectarse remotamente por RDP, Server Manager consola MMC y PowerShell.



Accede al vídeo

6.2. Ediciones de Windows

El ecosistema Windows no es tan extenso como el de las distribuciones de Linux, pero también puede dar lugar a confusión. Este tema se centrará en **Windows Server**, la

versión de sistema operativo dedicada a servidores. No se tratará, por tanto, ninguna edición de escritorio.

Dentro de la oferta de Windows Server, Microsoft publica versiones de actualización a menudo. De manera similar a otros sistemas operativos, estas versiones son soportadas durante unos pocos años. Es decir, Microsoft publicará parches de funcionalidad y seguridad durante el periodo de soporte, además de ofrecer asistencia técnica a sus clientes. Por ejemplo, el [ciclo de soporte](#) de Windows Server 2019 comenzó el 15/10/2019 y terminará el 12/1/2027. Este tema se centrará en Windows Server 2019, la versión más reciente, salvo cuando se indique lo contrario.

Windows Server 2019 está disponible en dos **ediciones**, también conocidas como SKU: Standard y Datacenter. Las ediciones se [diferencian](#) por su precio y las funcionalidades que soportan. La edición **Datacenter** está enfocada principalmente a entornos de virtualización y puede alojar un número ilimitado de máquinas virtuales (siempre que el servidor sea capaz de servir los requisitos de *hardware* virtual), mientras que la edición **Standard** solo puede ejecutar dos máquinas.

Además de algunas diferencias específicas de Hyper-V (el hipervisor de virtualización de Windows), Datacenter soporta *software defined networking* ([SDN](#), redes definidas por *software*), una funcionalidad esencial de cualquier entorno de nube privada o pública. Es habitual que Microsoft distribuya las ediciones en una misma imagen ISO, por lo que la elección de edición puede hacerse durante la instalación (ver Figura 1).

Ambas ediciones soportan, además, varios modos de ejecución:

- ▶ Escritorio o Server with Desktop Experience.
- ▶ Server Core.
- ▶ Nano Server.

El modo de escritorio es el más familiar, ya que convierte al servidor en un *host* similar a cualquier máquina Windows: tiene botón de inicio, ventanas y consolas de

configuración. El modo Core, sin embargo, no tiene escritorio. Mantiene las funcionalidades para soportar aplicaciones tradicionales y permite la activación de cualquier rol, pero no tiene el entorno de escritorio tradicional. Está pensando para ser administrado remotamente a través de la línea de comandos, con PowerShell o con consolas MMC remotas.

Hay otras diferencias, como que Server Core no incluye las herramientas de accesibilidad, soporte de audio o Internet Explorer. La Tabla 1 lista algunas de las diferencias en cuanto a disponibilidad de aplicaciones entre ambos modos.

	Modo de instalación	
	Core	Escritorio
Línea de comandos	Disponible	Disponible
PowerShell	Disponible	Disponible
.NET	Disponible	Disponible
Server Manager	No disponible	Disponible
perfmon.exe	No disponible	Disponible
mmc.exe	No disponible	Disponible
Event Viewer	No disponible	Disponible
Services.msc	No disponible	Disponible
Barra de tareas	No disponible	Disponible
Administrador de tareas	Disponible	Disponible
Internet Explorer / Microsoft Edge	No disponible	Disponible
Sistema de ayuda	No disponible	Disponible
Remote Desktop Services	Disponible	Disponible
mstsc.exe (cliente de Remote Desktop)	No disponible	Disponible

Tabla 1. [Diferencias](#) entre Server Core y Server with Desktop Experience. Fuente: elaboración propia.

Tanto el modo Core como el modo de escritorio se seleccionan durante la instalación del sistema (ver Figura 1). La consola del modo Core sí tiene, no obstante, ventanas. Por ejemplo, el administrador de tareas está disponible y tiene una interfaz gráfica, tal como muestra la Figura 2. Microsoft ha incluido un menú rápido en modo texto para la configuración inicial, `sconfig.cmd`. La sección A fondo incluye documentación en profundidad sobre otras herramientas de configuración de Server Core.

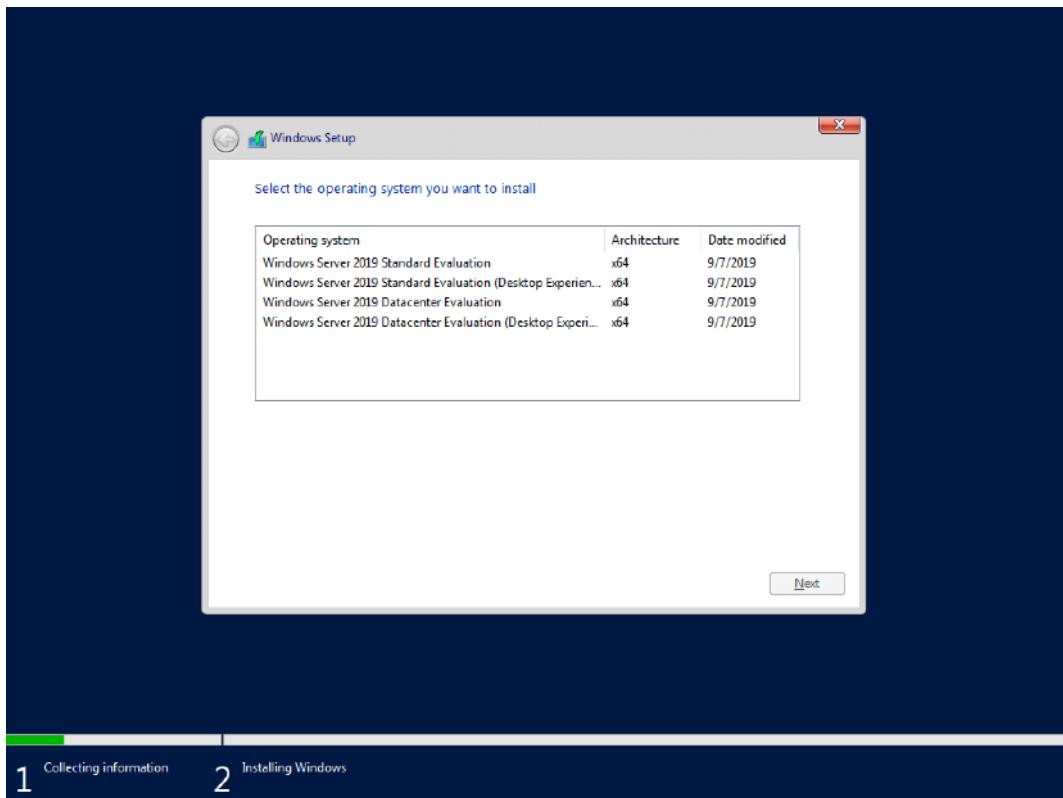


Figura 1. Elección de edición y modo en Windows Server 2019. Fuente: elaboración propia.

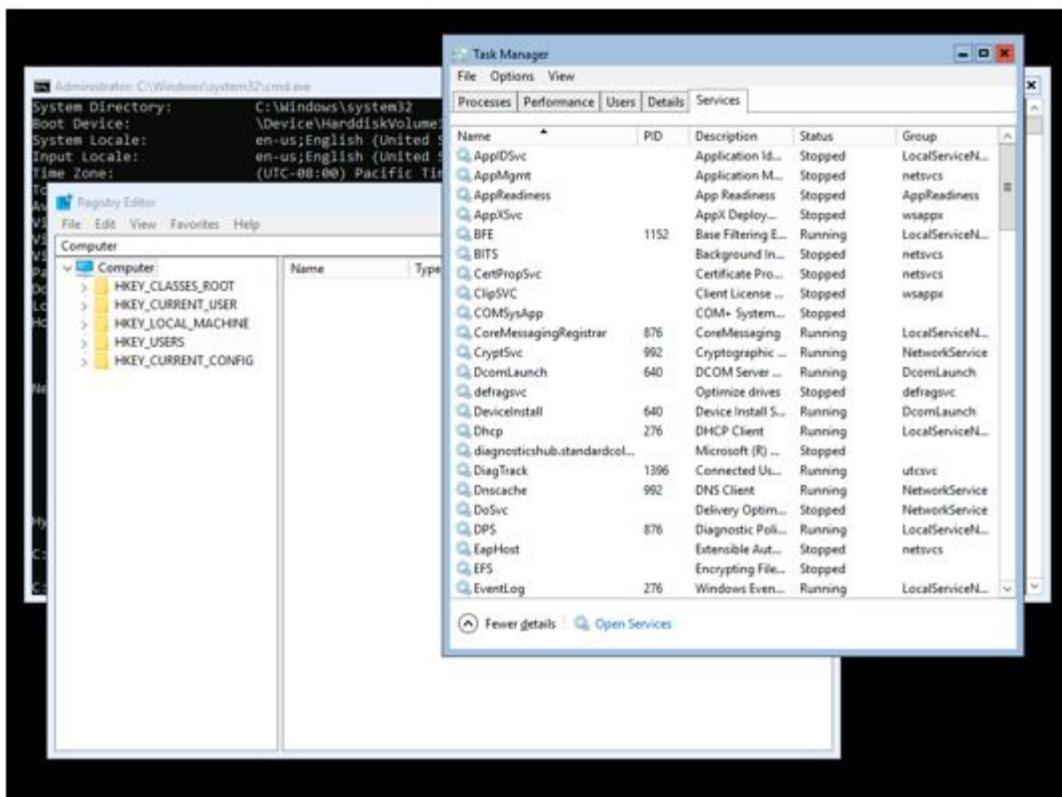


Figura 2. Interfaz del modo Server Core. Fuente: elaboración propia.

El tercer modo de ejecución mencionado anteriormente, **Nano Server**, no es un modo de instalación y, de hecho, no es posible instalarlo sobre *hardware* físico o virtual. Esta opción era posible con Windows Server [2016](#), pero no lo es con Windows Server [2019](#). Nano Server solo está disponible para ser ejecutado como un contenedor y, por tanto, Microsoft no ofrece un disco de instalación, sino una [imagen](#) base de contenedor. Nano Server es aún más limitado que Server Core, pero, al ser más ligero, está más indicado para entornos de virtualización.

6.3. Acceso remoto

Uno de los primeros pasos para administrar un servidor Windows, al igual que un servidor Linux, es poder acceder al mismo remotamente. Los siguientes apartados explicarán algunas de las principales [herramientas de administración remota](#):

- ▶ Escritorio remoto o RDP.
- ▶ PowerShell.
- ▶ Server Manager.
- ▶ Consola remota MMC.
- ▶ Remote Server Administration Tools o RSAT.
- ▶ Otros.

Escrivtorio remoto

Remote Desktop Protocol (RDP) es un protocolo propietario, desarrollado por Microsoft, que proporciona al usuario una interfaz gráfica para conectarse a otro ordenador a través de una conexión de red. El usuario necesita un cliente RDP mientras que el otro equipo debe tener habilitada la funcionalidad de acceso remoto. Esta se puede habilitar a través del panel de control (ver Figura 3) o con el comando sconfig.cmd en Server Core. RDP funciona sobre el puerto TCP 3389.

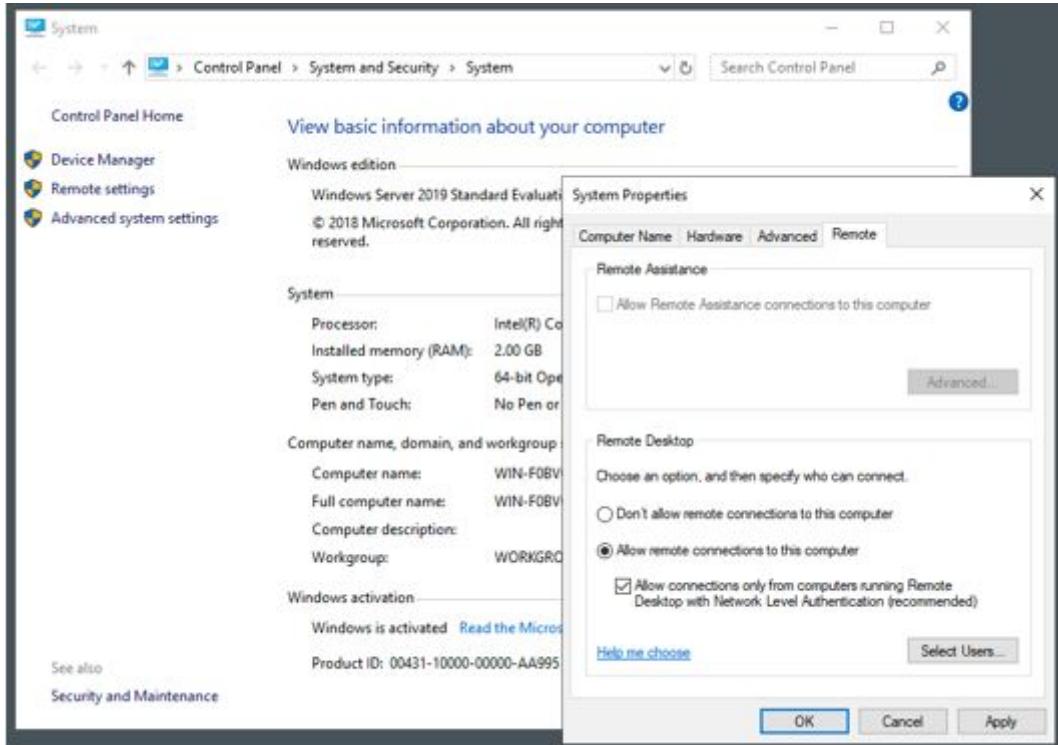


Figura 3. Funcionalidad de acceso remoto en Windows Server. Fuente: elaboración propia.

Server Core soporta RDP, aunque no tenga escritorio como tal. Tal como se comentó en el apartado anterior, Server Core no es una consola de texto, sino que tiene un entorno de ventanas. Una vez habilitado, una conexión de RDP es idéntica a una conexión a la consola virtual de una máquina virtual o a una pantalla física (ver Figura 4). Nano Server, sin embargo, no soporta RDP.

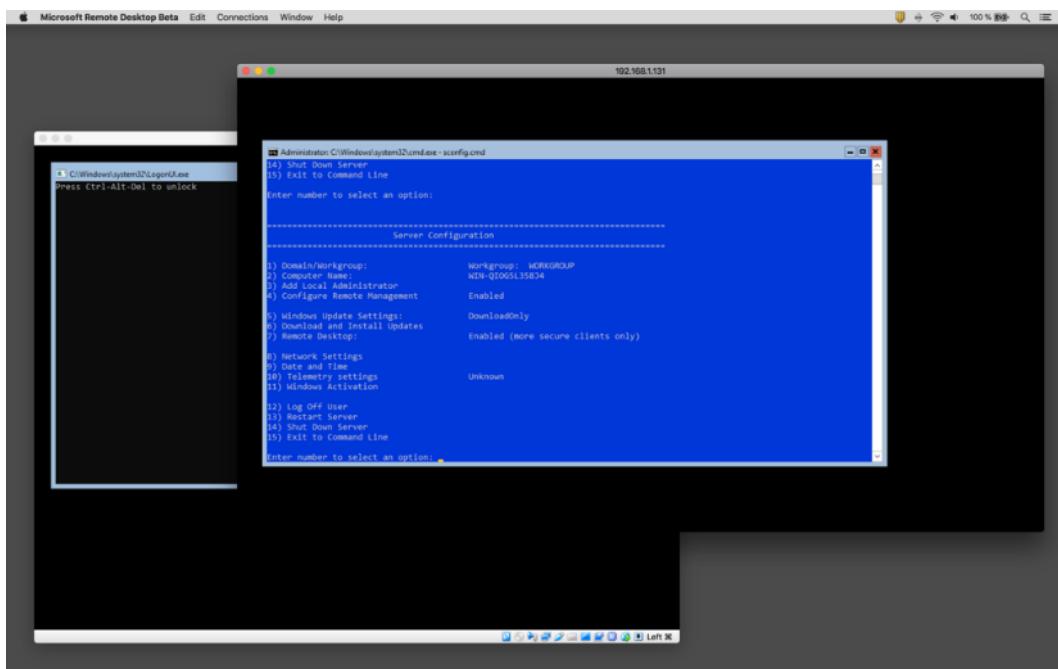


Figura 4. Conexión por RDP a un servidor Windows 2019 Server Core con el cliente RDP en MacOS. Fuente: elaboración propia.

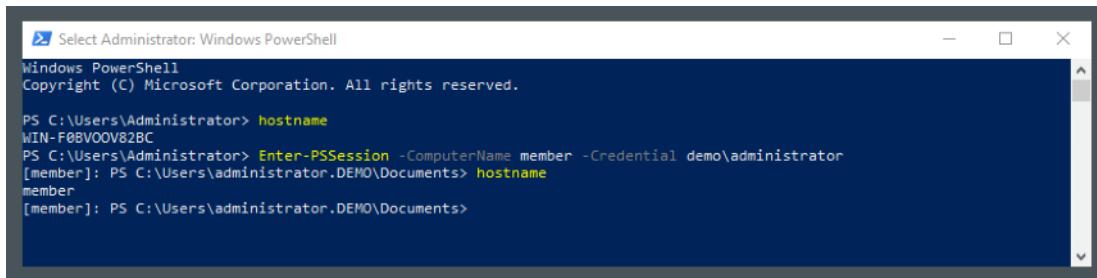
PowerShell

La **consola nativa de Windows**, PowerShell, permite ejecutar comandos al iniciar sesiones de consola en *hosts* remotos. Está disponible tanto en el servidor con escritorio como en Server Core y puede habilitarse en Nano Server si se instala PowerShell Core. Este apartado se limitará a la configuración para la ejecución remota.

Es necesario habilitar la invocación remota en el *host* de destino con Enable-PSRemoting -Force. A partir de ese momento, el *host* aceptará llamadas de PowerShell de otros equipos del dominio. Por ejemplo, el siguiente comando devolverá el contenido de la unidad C:

```
Invoke-Command -ComputerName 192.168.1.130 -ScriptBlock { Get-ChildItem C:\} -credential USERNAME
```

Se podría iniciar una sesión remota con Enter-PSSession y ejecutar los comandos de forma interactiva. Por ejemplo, la Figura 5 muestra un inicio de sesión remota entre dos equipos que forman parte del mismo dominio.

A screenshot of a Windows PowerShell window titled "Select Administrator: Windows PowerShell". The window shows the command "Enter-PSSession -ComputerName member -Credential demo\administrator" being run from a local machine named "WIN-F0BV0OV8BC". The session has been established, and the prompt now shows the local machine name "member" followed by the command "hostname".

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

PS C:\Users\Administrator> hostname
WIN-F0BV0OV8BC
PS C:\Users\Administrator> Enter-PSSession -ComputerName member -Credential demo\administrator
[member]: PS C:\Users\administrator.DEMO\Documents> hostname
member
[member]: PS C:\Users\administrator.DEMO\Documents>
```

Figura 5. Sesión remota con PowerShell. Fuente: elaboración propia.

Server Manager

La herramienta de Administración de Servidor, o Server Manager, ofrece una interfaz gráfica para **ejecutar tareas de configuración**, tanto en el equipo local como en equipos remotos. Ofrece acceso a consolas de administración de roles de Windows, como DNS o Directorio Activo, desde un único punto, instalación de roles, escritorio remoto, etc. La Figura 6 muestra una instancia de Server Manager con dos equipos y las opciones que ofrece para administración remota. Server Manager permite administrar tanto servidores con escritorio como Server Core y Nano Server.

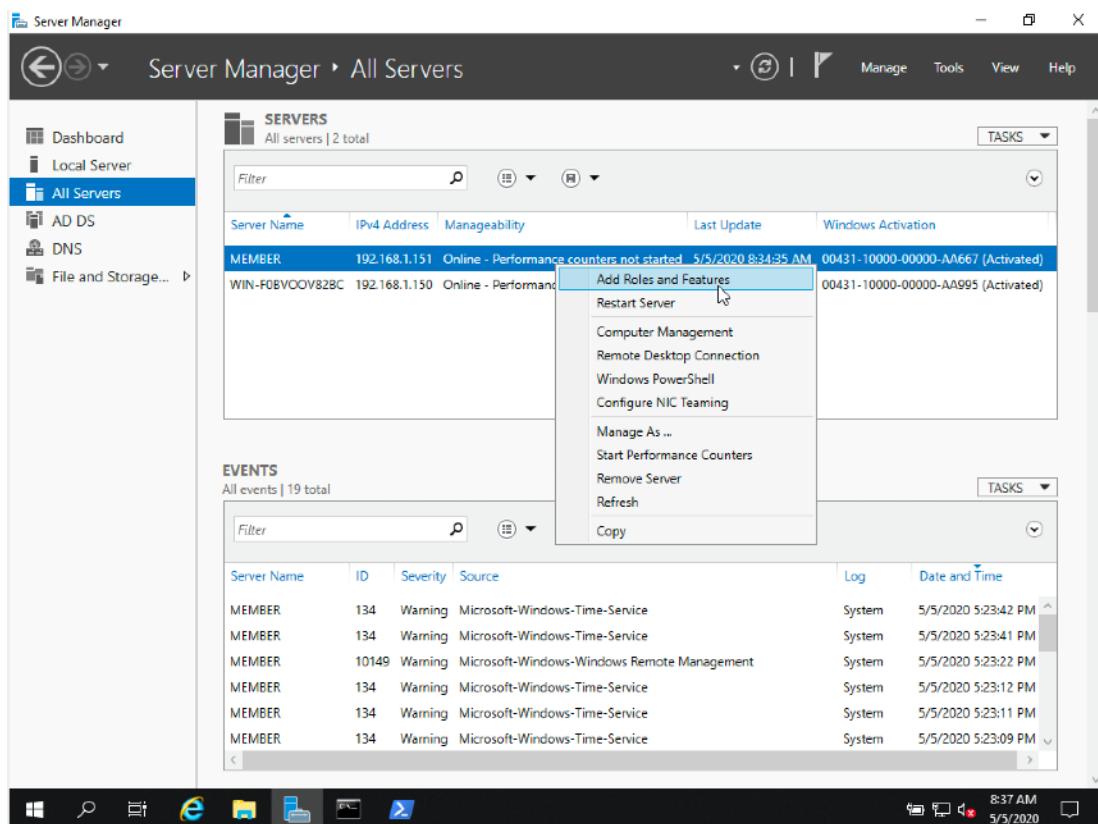


Figura 6. Server Manager con dos equipos. Fuente: elaboración propia.

Consola remota MMC

La consola tradicional de administración de Windows, MMC (Microsoft Management Console), es otra herramienta gráfica de configuración. Mientras que Server Manager es una herramienta centralizada, MMC ofrece múltiples *snapins* para cada funcionalidad y solo permite administrar un equipo con cada *snapin*. De hecho, parte de la funcionalidad de Server Manager es abrir la consola MMC con el *snapin* correcto en un equipo concreto, ahorrando tiempo al administrador. MMC soporta servidores con escritorio, Server Core y Nano Server.

La Figura 7 muestra dos consolas MMC, una con el *snapin* de DNS conectada al equipo local y otra con el *snapin* de administración de discos conectada a un equipo remoto.

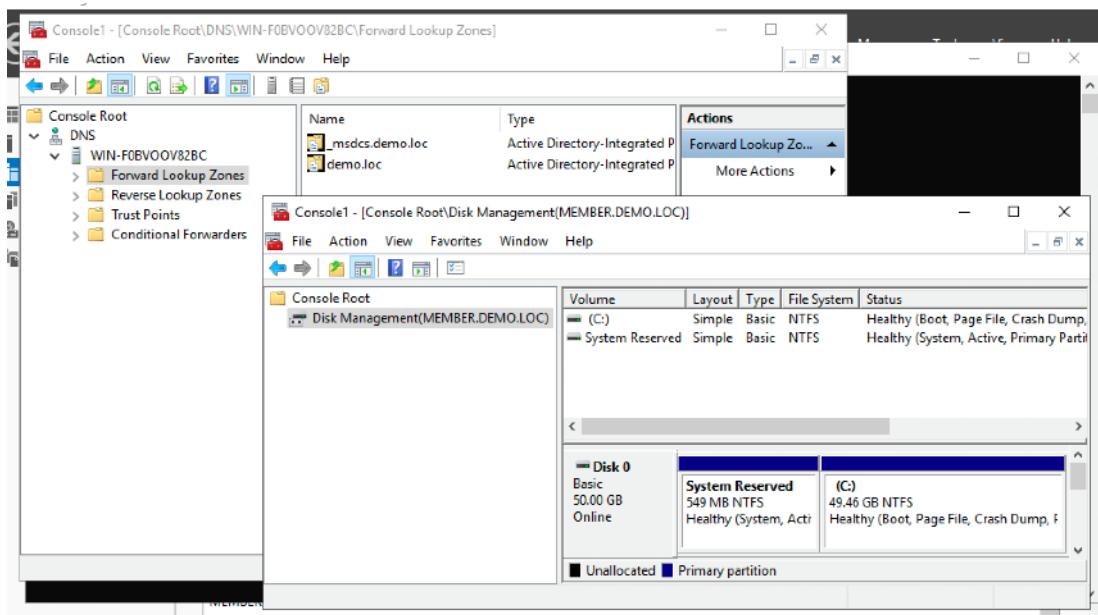


Figura 7. Consola MMC con los *snapsins* de DNS y Disk Management. Fuente: elaboración propia.

Remote server administration tools

[RSAT](#) no es más que una colección de herramientas de administración remotas, listas para ser instaladas en un equipo Windows cliente (por ejemplo, Windows 10) con un único paquete. Incluye Server Manager, la consola MMC con múltiples *snapsins* y *commandlets* de PowerShell que solo las versiones de Windows Server incorporan por defecto.

Otros

Al igual que en Linux, la oferta de sistemas de administración remota es muy amplia. Por ejemplo, hay versiones de VNC y TeamViewer disponibles para Windows. Microsoft incluyó [OpenSSH](#) en Windows Server 2019 y en Windows 10, lo que añade otra herramienta al arsenal.

6.4. Directorio Activo

Directorio Activo, o AD (Active Directory), es un servicio de directorio para redes de equipos Windows. Cualquier versión de Windows Server puede actuar como controlador de dominio, desde Windows Server 2000. AD se usa para **tareas de administración centralizada**, aparte de integrar otras funcionalidades, como servicios de certificados.

Un controlador de dominio es un servidor Windows que ejecuta el servicio Active Directory Domain Service. **Autentica y autoriza** todos los usuarios y equipos de un dominio y les aplica políticas de seguridad. Por ejemplo, cuando un usuario inicia sesión en un ordenador que es parte de un dominio Windows, un controlador de dominio, y no el equipo local, comprueba que la contraseña sea válida.

A nivel de protocolos, AD se sirve de múltiples tecnologías:

- ▶ DNS para localizar servicios (gracias a entradas SRV) y para identificar tanto dominios como los equipos miembros del dominio.
- ▶ LDAP es el servicio de directorio como tal. Sirve de base de datos para alojar información de las cuentas de máquinas y de usuario. Es extensible, por lo que sistemas como Exchange pueden alojar datos adicionales de cada usuario, como los datos de la cuenta de correo, sin necesidad de crear otro repositorio de datos.
- ▶ Kerberos es el sistema de autenticación y autorización entre equipos.

AD funciona esencialmente como una base de datos administrativa. Usa una estructura jerárquica formada por objetos, dominio, árboles y bosques (DNS Stuff, 2019).

Objetos

Los objetos son los **elementos básicos del dominio**. Representan los ordenadores cliente (por ejemplo, los equipos ofimáticos de escritorio, que actúan como clientes de AD), los servidores de aplicaciones (que, a efectos del directorio, son también clientes de AD), carpetas compartidas y cuentas de usuario. Cada objeto tiene un identificador único y un conjunto de atributos. Por ejemplo, la Figura 8 muestra algunos de los campos básicos de una cuenta de usuario, aunque en el resto de las pestañas hay muchos más.

Los objetos se pueden organizar en objetos **contenedores**, como **carpetas y unidades organizativas**, o OU. Las carpetas son similares a las de un sistema de ficheros, pero las unidades organizativas permiten, además, aplicar políticas de seguridad a los objetos de la OU y delegar la administración a otros usuarios.

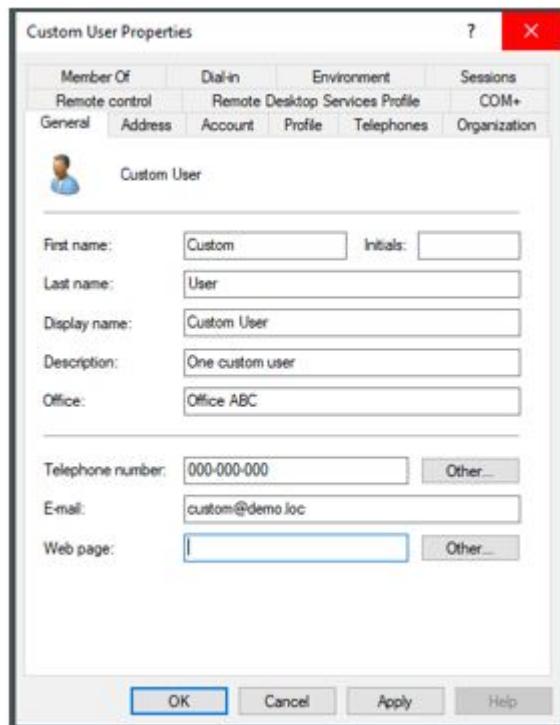


Figura 8. Propiedades de usuario de AD. Fuente: elaboración propia.

Dominios

Un dominio es un área de la red con una única base de datos de autenticación. A efectos prácticos, sirve para establecer límites administrativos entre diferentes grupos de red. Cualquier objeto pertenece a un único dominio y los objetos de un mismo dominio pueden estar dispersos en diferentes ubicaciones físicas. Por ejemplo, una organización puede establecer un dominio para cada departamento, pero en una misma oficina pueden convivir usuarios y equipos de departamentos diferentes sin que eso rompa la estructura del dominio.

Cada dominio tendrá al menos un **controlador de dominio**, o DC, que actúa como autoridad de aquel; es decir, es responsable de los permisos de los objetos, la autenticación y las modificaciones de objetos. Es habitual que un dominio disponga de más de un DC. Esta disposición es posible porque los DC pueden trabajar de manera distribuida y es necesaria porque la caída del único DC de un dominio colapsaría todas las máquinas del dominio, ya que nadie podría iniciar sesión ni acceder a carpetas compartidas. Los controladores de dominio suelen alojar el servicio DNS, que también puede funcionar en modo distribuido.

Cada dominio tiene una estructura de **árbol de contenedores**. Estos contenedores pueden ser carpetas normales o unidades organizativas. La Figura 9 muestra el árbol de carpetas de un dominio y el contenido de una de las carpetas.

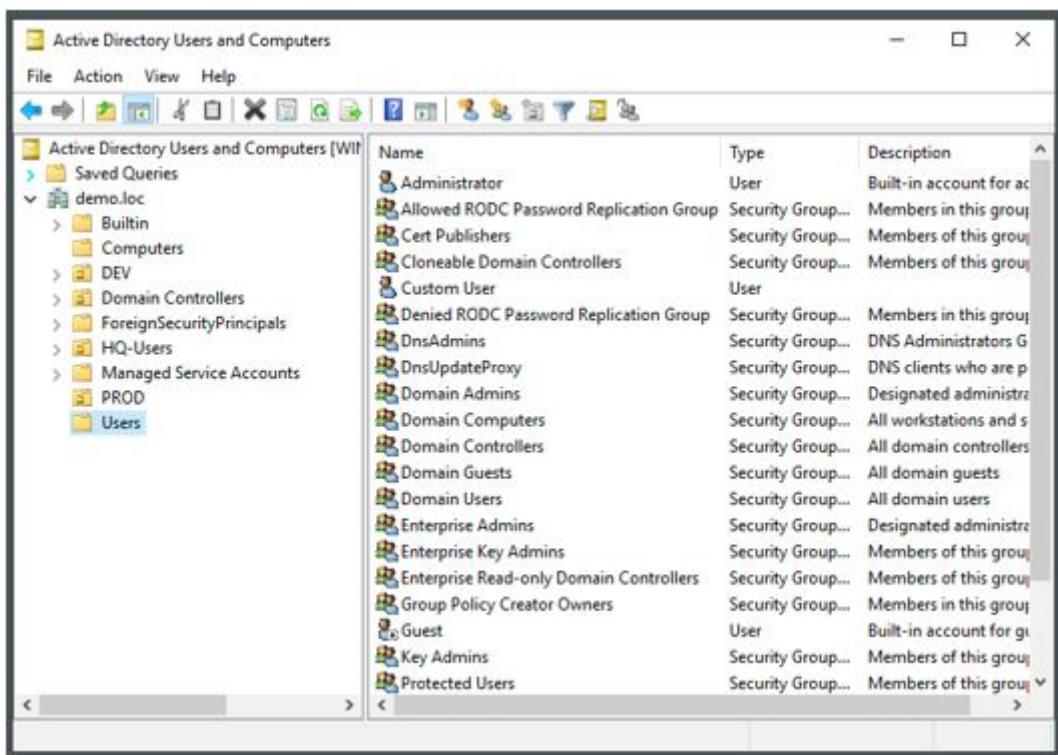


Figura 9. Estructura de árbol del dominio demo.loc. Fuente: elaboración propia.

Los dominios se identifican con un nombre DNS. De hecho, la creación de un dominio requiere la presencia de un servicio DNS asociado, bien externo o integrado en el propio dominio. Al ser un nombre DNS, las cuentas de máquina reciben un **nombre de dominio completo** (o FQDN, *fully qualified domain name*) en el que se une el nombre del *host* con el nombre del dominio.

Un equipo con nombre *member* en el dominio demo.loc tendrá un registro DNS member.demo.loc. La integración de AD con DNS es tan fuerte que los dominios siguen una estructura de árbol, de manera que un dominio padre puede ser demo.loc y un dominio hijo podría ser, a su vez, child.demo.loc. Cada dominio formará un **límite administrativo**, pero habrá ciertas interacciones permitidas entre ellos. Los dominios AD puede tener el nombre de dominio de Internet de una empresa, ya que es un nombre DNS válido, pero no es estrictamente necesario.

Bosques

Las empresas con cientos o miles de usuarios pueden disponer de varios árboles de directorios. Los administradores podrán organizar los árboles en bosques. Los bosques son el límite administrativo y de seguridad más amplio en una estructura AD.

Una organización puede decidir tener no ya varios dominios, sino varios bosques. Esta situación es habitual, por ejemplo, cuando hay uniones y adquisiciones de organizaciones. En estos casos, es posible conectar los bosques con una **relación de confianza**. Estas relaciones extienden la accesibilidad de los recursos de los dominios, de manera que centraliza la administración a nivel lógico.

Sitios

Los sitios de AD sirven para administrar organizaciones que tienen sucursales en diferentes ubicaciones geográficas, pero que caen bajo el mismo dominio. Los sitios son agrupaciones físicas de subredes IP bien conectadas, que se utilizan para replicar de manera eficiente la información entre los controladores de dominio (DC). Permiten ejercer control sobre el tráfico de replicación y el proceso de autenticación. Cuando hay DC disponibles en un sitio dado, los servicios de los equipos miembro pueden dirigir a estos las consultas de inicio de sesión y las búsquedas en el directorio. Los sitios también sirven para desplegar directivas de grupo específicas a nivel geográfico (las OU aplican las directivas a nivel lógico o departamental).

Sincronización de AD

Los objetos de un dominio se mantienen en una base de datos replicada y distribuida. Cada controlador de dominio contiene una copia de todos los objetos del dominio y puede servir prácticamente para todas las operaciones de AD. Hay ciertos roles que solo pueden estar asignados a un único DC en cada dominio. Si el DC con uno de esos roles, denominado *operation master*, deja de estar disponible, las tareas de ese rol no se pueden ejecutar. Estas tareas no son tan habituales como un inicio de sesión o un

cambio de contraseña, por lo que admiten cierto *downtime*, sin impactar negativamente en el funcionamiento del dominio.

Cualquier cambio en un objeto realizado en un controlador de dominio se transferirá automáticamente a las réplicas en los otros DC. Como la topología de dominios es independiente de la topología de red, es habitual que los DC de un dominio estén distribuidos geográficamente para proporcionar una mejor resiliencia.

Niveles funcionales

Todas las versiones de Windows Server usan un concepto llamado **nivel funcional de bosque y de dominio** (Panek, 2018). El nivel funcional se escoge en el momento de la creación y determina las funcionalidades disponibles. En una instalación de cero, es habitual escoger el nivel más alto, pero en el momento en que hay controladores de dominio de diversas versiones de Windows Server (una situación muy habitual, especialmente en situaciones de actualización a una versión reciente), la respuesta no es tan sencilla.

Windows Server 2019 no añade ningún nivel funcional nuevo, sino que mantiene los niveles de Windows Server 2016. Este nivel funcional añade funcionalidades, como el soporte de NTLM para usuarios restringidos a dispositivos específicos del dominio, pero tiene la limitación de que todos los controladores del dominio deben ser Windows Server 2016 o Windows Server 2019. Es habitual tener un dominio en un nivel funcional dado (Windows Server 2012, por ejemplo) con todos los DC en una versión dada (Windows Server 2012 R2, por ejemplo) y que, durante un proceso de actualización, se sustituyan los DC por otros nuevos con Windows Server 2019. Una vez todos los DC antiguos han sido sustituidos, el administrador puede elevar el nivel funcional y hacer uso de las nuevas funcionalidades.

6.5. Creación de un dominio

Un servidor Windows Server se convierte en controlador de dominio mediante la activación del rol **Active Directory Domain Services**. Este apartado ejemplifica cómo activar este rol en un equipo recién instalado y cómo crear el primer dominio en un bosque nuevo.

- ▶ El primer paso es configurar la tarjeta de red del servidor que se convertirá en controlador del dominio. La instalación de AD es muy dependiente del despliegue de DNS subyacente, por lo que el servidor suele tener una IP estática (ver Figura 10). No es un requisito del proceso de instalación, pero es muy habitual en instalaciones empresariales. El servidor DNS de la subred se configura de manera estática, también, pero es habitual, una vez creado el dominio, configurar la propia IP del equipo como servidor DNS, ya que el servicio DNS de AD se encargará de resolver nombres tanto del dominio de AD como de dominios externos.
- ▶ También conviene fijar el nombre de la máquina antes de promocionarla a controlador de dominio. En este ejemplo, el *hostname* será dc1.

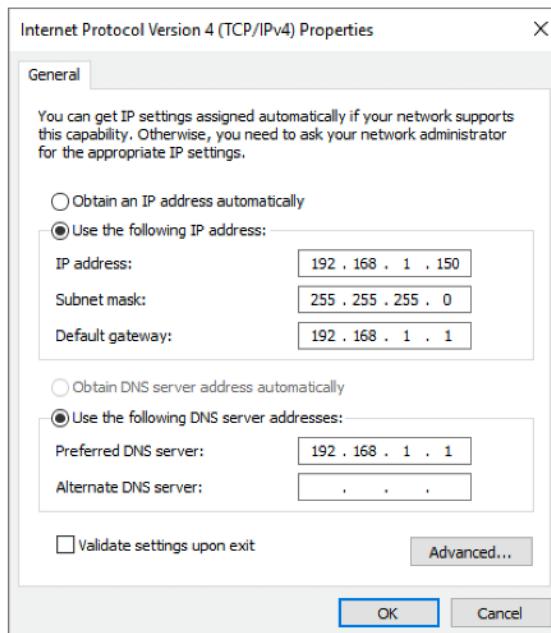


Figura 10. Tarjeta de red con IP estática. Fuente: elaboración propia.

- ▶ El siguiente paso consiste en añadir el rol de Active Directory Domain Services. Esto se puede llevar a cabo desde el Server Manager, en Add roles and features. Tras seleccionar el equipo local, hay que marcar el rol Active Directory Domain Services y aceptar la instalación de herramientas adicionales, como el la Figura 11. En el resto de los pasos de la instalación, se pueden aceptar los valores por defecto, ya que la configuración del dominio se lleva a cabo en un cuadro de diálogo diferente, tras la instalación.

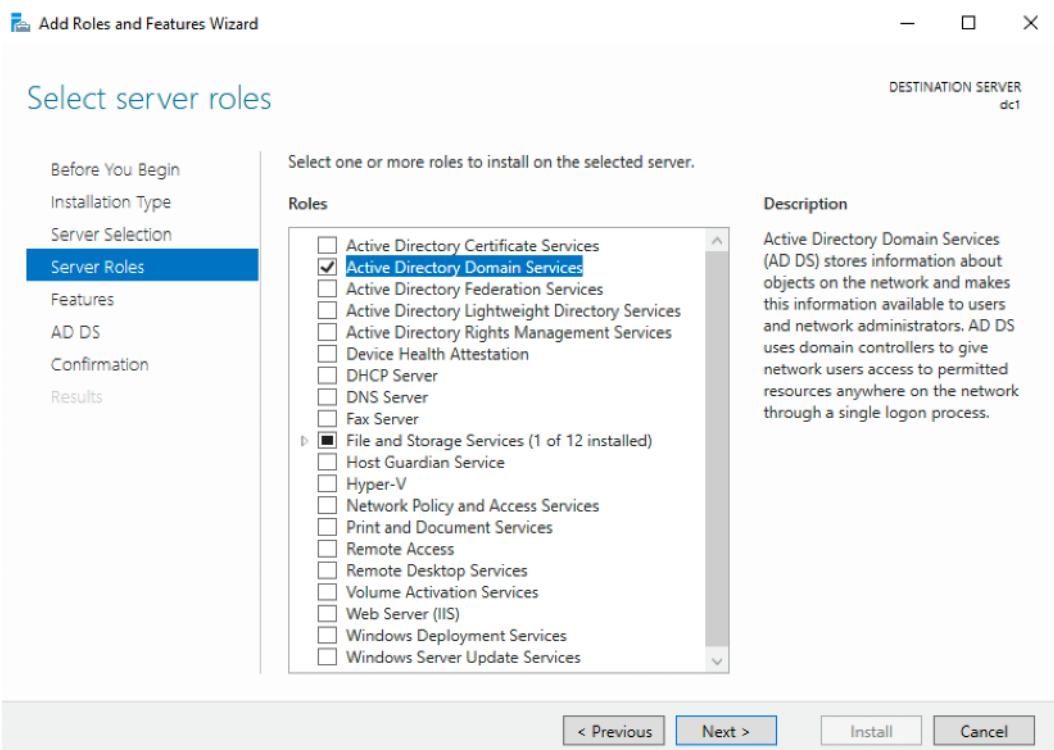


Figura 11. Selección de rol Active Directory Domain Services. Fuente: elaboración propia.

- ▶ Una vez terminada la instalación del rol, Server Manager muestra una opción para promocionar el servidor a controlador de dominio. En versiones anteriores, este proceso se arranca con el comando `dcpromo`, pero ya no está disponible en Windows Server 2019.

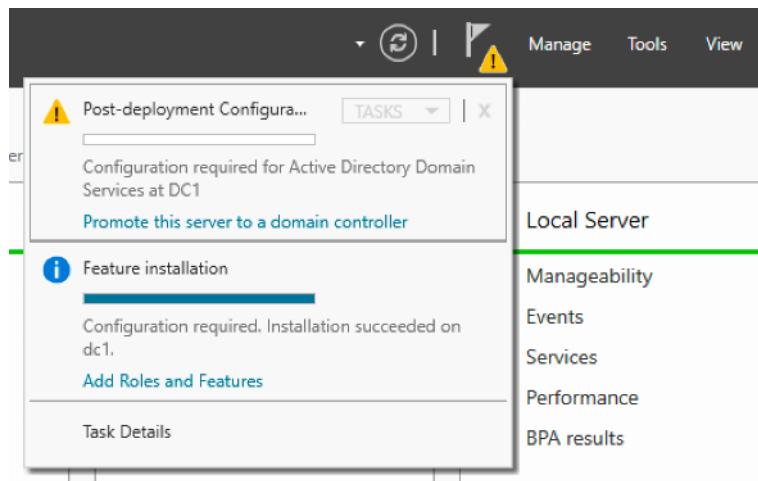


Figura 12. Mensaje de promoción a controlador de dominio. Fuente: elaboración propia.

- ▶ La primera página del diálogo de promoción es dónde promocionar el DC: en un dominio existente, en un dominio nuevo en bosque existente o en un nuevo dominio en un nuevo bosque. En la primera opción, solo hay que seleccionar el dominio en el que ubicar el DC. En la segunda, hay que indicar el dominio padre, si el dominio pertenecerá a un árbol existente, o simplemente el nombre del bosque, si el nuevo dominio será la raíz de un árbol nuevo. En la última opción, la que se seguirá en este ejemplo, es necesario indicar el nombre del bosque y el nombre del dominio, que será el dominio raíz en el primer árbol (ver Figura 13).

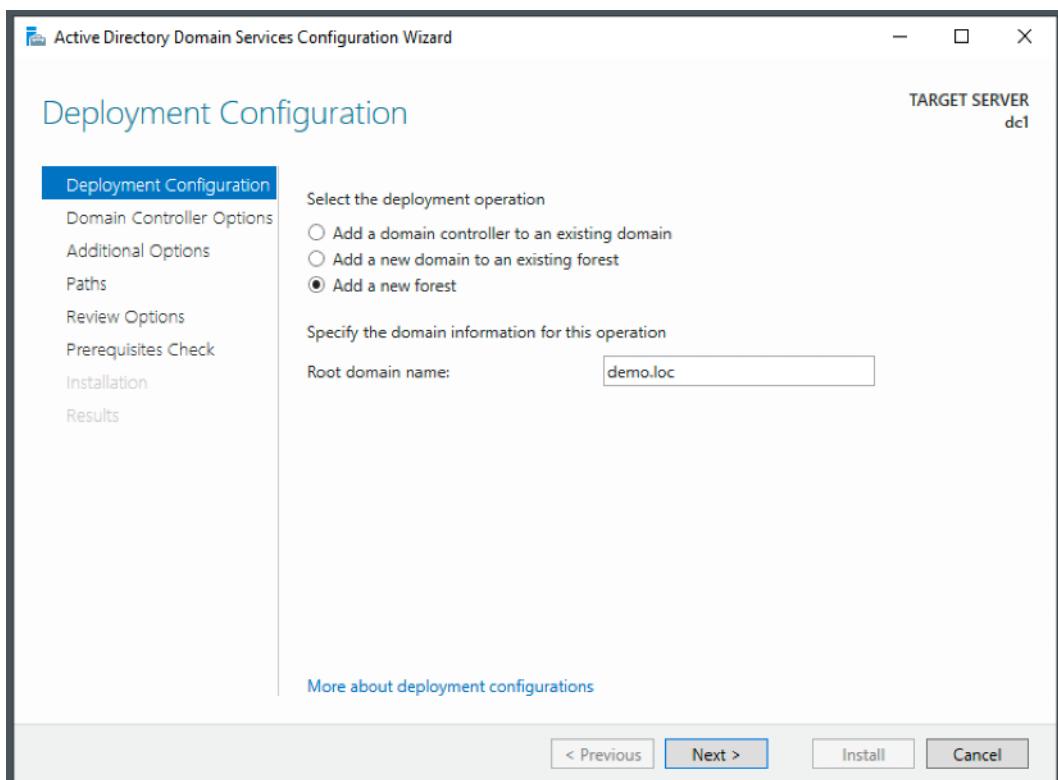


Figura 13. Controlador de dominio en un nuevo bosque. Fuente: elaboración propia.

- ▶ A continuación, se seleccionan los niveles funcionales del dominio y del bosque. Este ejemplo despliega un dominio de cero, así que es seguro escoger el nivel más alto, pero, en un entorno con versiones antiguas de DC, habría que revisar las matrices de compatibilidad (ver Figura 14).

- ▶ El diálogo ofrece varios servicios disponibles para el controlador de dominio:
 - DNS. Es habitual, pero no estrictamente necesario, que un DC sea un servidor DNS. Con un servicio DNS integrado, la gestión de nombres se simplifica, ya que cualquier equipo unido al dominio será dado de alta en la zona DNS automáticamente.
 - Catálogo Global. El servicio de GC acelera las búsquedas de recursos en AD, gracias a que el equipo almacena toda la información del esquema y un subconjunto de los atributos de los objetos de todos los dominios del bosque (Panek, 2018). Cada dominio debe tener al menos uno, pero suele haber más.
 - Controlador de dominio de solo lectura (RODC). Estos DC suelen desplegarse en oficinas remotas con baja seguridad, donde es necesario desplegar un DC

para acelerar los inicios de sesión, pero existe riesgo de que un ataque altere los datos del DC.

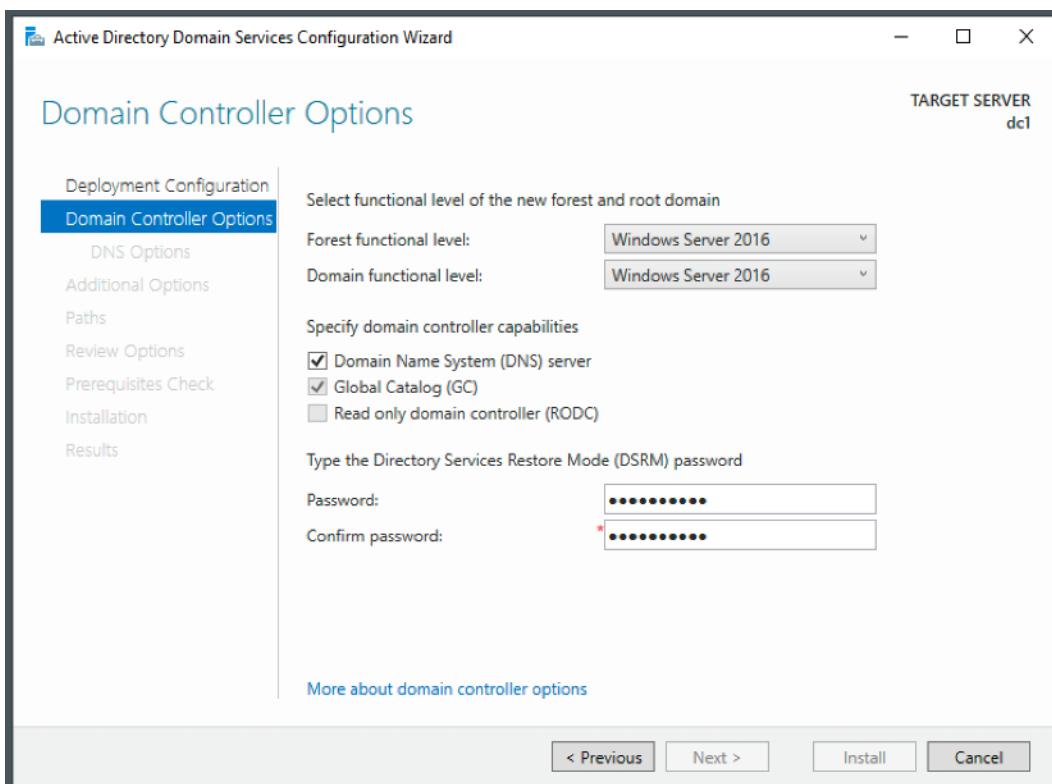


Figura 14. Niveles funcionales y servicios adicionales. Fuente: elaboración propia.

- ▶ Los siguientes pasos solicitan detalles sobre la configuración DNS (si el DC va a actuar como tal) y del dominio NetBIOS. Los nombres NetBIOS son herencia de las versiones más antiguas de Windows Server (concretamente, de Windows NT), que Microsoft sigue manteniendo por compatibilidad. También es posible cambiar las rutas por defecto de la ubicación de la base de datos de AD. En este caso, no es necesario cambiar estas opciones.
- ▶ El diálogo de instalación termina comprobando que todas las opciones son válidas. Si todo va bien, el proceso termina cerrando la sesión del usuario. Un controlador de dominio no tiene cuentas locales, ya que solo es posible iniciar sesión con cuentas de dominio (en un equipo miembro es posible iniciar sesión tanto con cuentas locales como con cuentas de dominio).

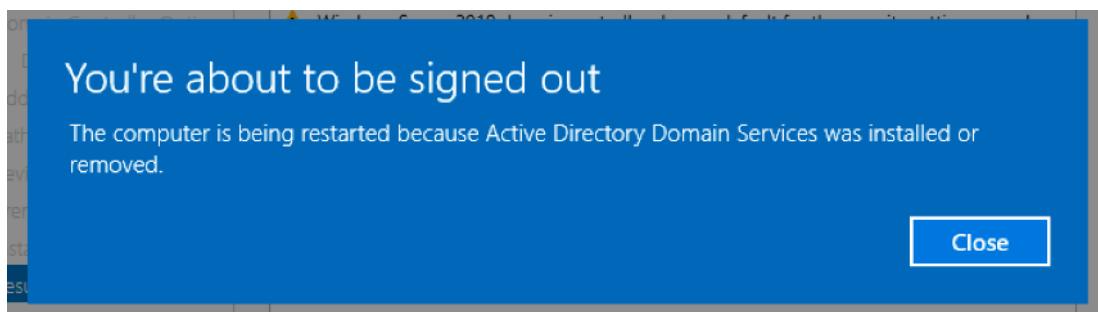


Figura 15. Fin de la promoción del servidor a controlador de dominio. Fuente: elaboración propia.

Tras el reinicio, el equipo ya actuará como DC del nuevo dominio. Los equipos de la red podrán hacerle peticiones siempre que puedan localizarlo por DNS. La Figura 16 muestra algunos de los registros SRV del nuevo dominio. Estos registros no son utilizados para resolver nombres de equipo, sino para localizar servicios. Por ejemplo, la entrada _ldap de la imagen indica que el equipo dc1.demo.loc ofrece el servicio LDAP en el dominio demo.loc. Cuando se solicita, por ejemplo, la unión de un equipo cliente a un dominio, Windows consultará estos registros para localizar el controlador de dominio al que dirigir las peticiones.

The screenshot shows the Windows DNS Manager interface. The left pane displays a tree view of DNS zones under 'dc1.demo.loc'. The 'Forward Lookup Zones' section contains several sub-zones like '_msdcs.demo.loc', 'dc', '_sites', '_tcp', 'domains', and '51221d31-9a1e-4'. The right pane lists 'Service Location (SRV)' records:

Name	Type	Data	Timestamp
_kerberos	Service Location (SRV)	[0][100][88] dc1.demo.loc.	5/5/2020 3:00:00 PM
_ldap	Service Location (SRV)	[0][100][389] dc1.demo.loc.	5/5/2020 3:00:00 PM

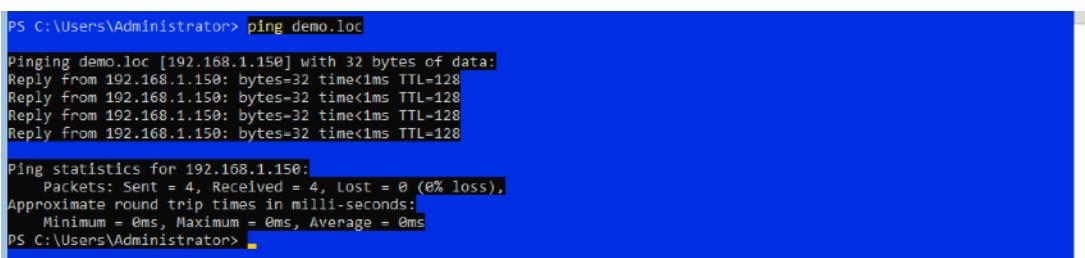
Figura 16. Registros SRV de un Directorio Activo. Fuente: elaboración propia.

Unión de equipos al dominio

Mientras que promocionar controladores de dominio es una tarea relativamente poco frecuente, la **unión de equipos al dominio** ocurre a menudo: cada nuevo

servidor Windows y cada nuevo equipo de cliente debe ser unido al dominio para facilitar los inicios de sesión, el control de políticas de seguridad, etc.

En un equipo con Server Core, por ejemplo, se puede usar el menú sconfig.cmd para unir el equipo al dominio. Una primera comprobación debe ser que el sistema pueda resolver correctamente el nombre del dominio (ver Figura 17).

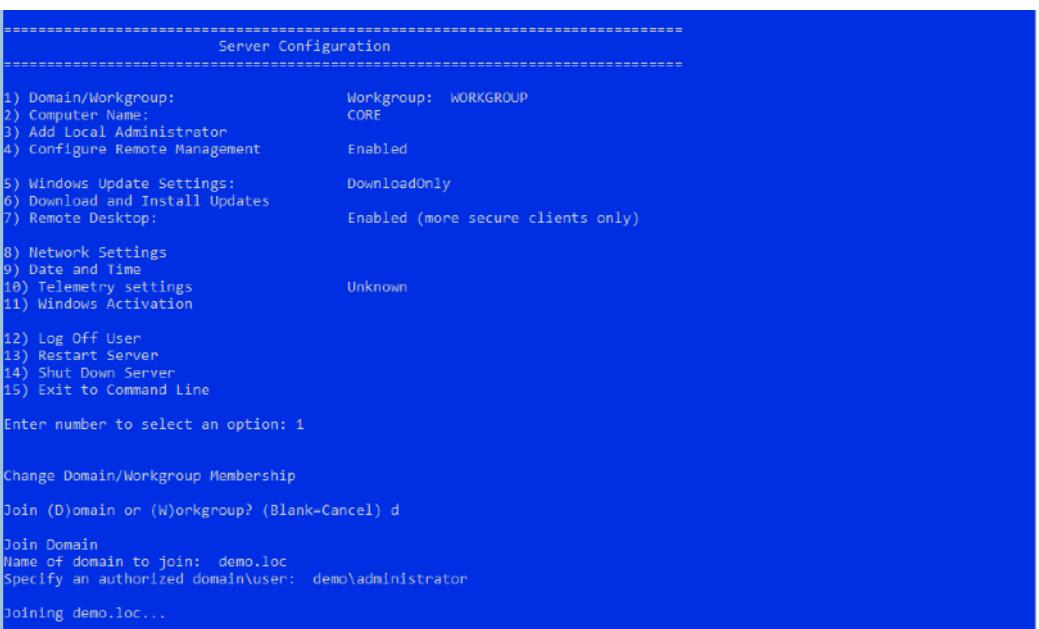


```
PS C:\Users\Administrator> ping demo.loc
Pinging demo.loc [192.168.1.150] with 32 bytes of data:
Reply from 192.168.1.150: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.1.150:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 0ms, Maximum = 0ms, Average = 0ms
PS C:\Users\Administrator>
```

Figura 17. Resolución del nombre del dominio con ping. Fuente: elaboración propia.

Una vez comprobado, el menú de sconfig solicita el nombre del dominio y las credenciales de un administrador (ver Figura 18).



```
=====
Server Configuration
=====

1) Domain/Workgroup:          Workgroup: WORKGROUP
2) Computer Name:             CORE
3) Add Local Administrator
4) Configure Remote Management: Enabled
5) Windows Update Settings:   DownloadOnly
6) Download and Install Updates
7) Remote Desktop:            Enabled (more secure clients only)

8) Network Settings
9) Date and Time
10) Telemetry settings       Unknown
11) Windows Activation

12) Log Off User
13) Restart Server
14) Shut Down Server
15) Exit to Command Line

Enter number to select an option: 1

Change Domain/Workgroup Membership
Join (D)omain or (W)orkgroup? (Blank=Cancel) d

Join Domain
Name of domain to join: demo.loc
Specify an authorized domain\user: demo\administrator
Joining demo.loc...
```

Figura 18. Unión de Server Core a un dominio. Fuente: elaboración propia.

Tras un reinicio, el equipo será un servidor «miembro» del dominio, con todo lo que ello conlleva. Entre otras cosas, se habrá creado una cuenta de máquina en el directorio con el nombre del equipo, como muestra la Figura 19.

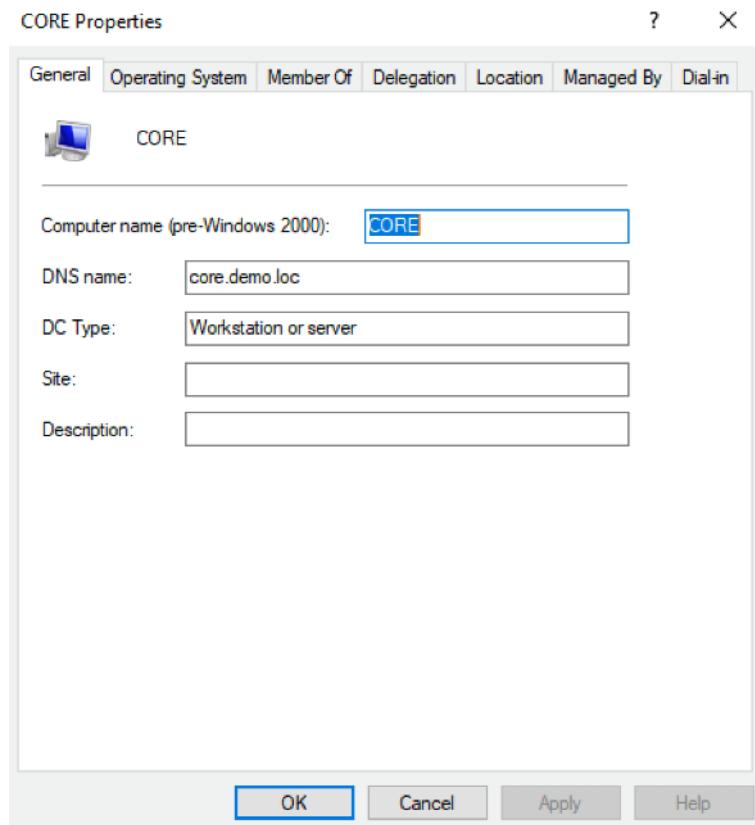


Figura 19. Detalles de cuenta de máquina. Fuente: elaboración propia.

6.6. Referencias bibliográficas

DNS Stuff. (2019, diciembre 16). *Active Directory Forest and Domain Guide 2020 + Best Tools*. <https://www.dnsstuff.com/active-directory-forest>

Docker Hub. (s. f.). *Nano Server*. https://hub.docker.com/_/microsoft-windows-nanoserver?tab=description

Lynn, S. (2013). *Windows Server 2012: Up and Running*. O'Reilly Media Inc.

Microsoft. (2018, febrero 20). *What is the Server Core installation option in Windows Server?* <https://docs.microsoft.com/en-us/windows-server/administration/server-core/what-is-server-core>

Microsoft. (2018, agosto 9). *SDN in Windows Server overview.* <https://docs.microsoft.com/en-us/windows-server/networking/sdn/software-defined-networking>

Microsoft. (2019, enero 7). *OpenSSH in Windows.* https://docs.microsoft.com/en-us/windows-server/administration/openssh/openssh_overview

Microsoft. (2019, mayo 5). *Install Nano Server.* <https://docs.microsoft.com/en-us/windows-server/get-started/nano-in-semi-annual-channel>

Microsoft. (2019, mayo 21). *Changes to Nano Server in Windows Server Semi-Annual Channel.* <https://docs.microsoft.com/en-us/windows-server/get-started/nano-in-semi-annual-channel>

Microsoft. (2019, julio 23). *Manage a Server Core server.* <https://docs.microsoft.com/en-us/windows-server/administration/server-core/server-core-manage#managing-with-microsoft-management-console>

Microsoft. (2020, agosto 25). *Forest and Domain Functional Levels.* <https://docs.microsoft.com/en-us/windows-server/identity/ad-ds/active-directory-functional-levels>

Microsoft. (2020, septiembre 9). *Remote Server Administration Tools.* <https://docs.microsoft.com/en-us/windows-server/remote/remote-server-administration-tools>

Microsoft. (2020, octubre 12). *Comparison of Standard and Datacenter editions of Windows Server 2019*. <https://docs.microsoft.com/en-gb/windows-server/get-started-19/editions-comparison-19>

Microsoft. (s. f.). *Buscar información sobre el ciclo de vida de los productos y servicios*. <https://docs.microsoft.com/es-es/lifecycle/products/?alpha=windows%20server%202016>

Panek, W. (2018). *MCSA Windows Server 2016 Complete Study Guide*. Sybex.

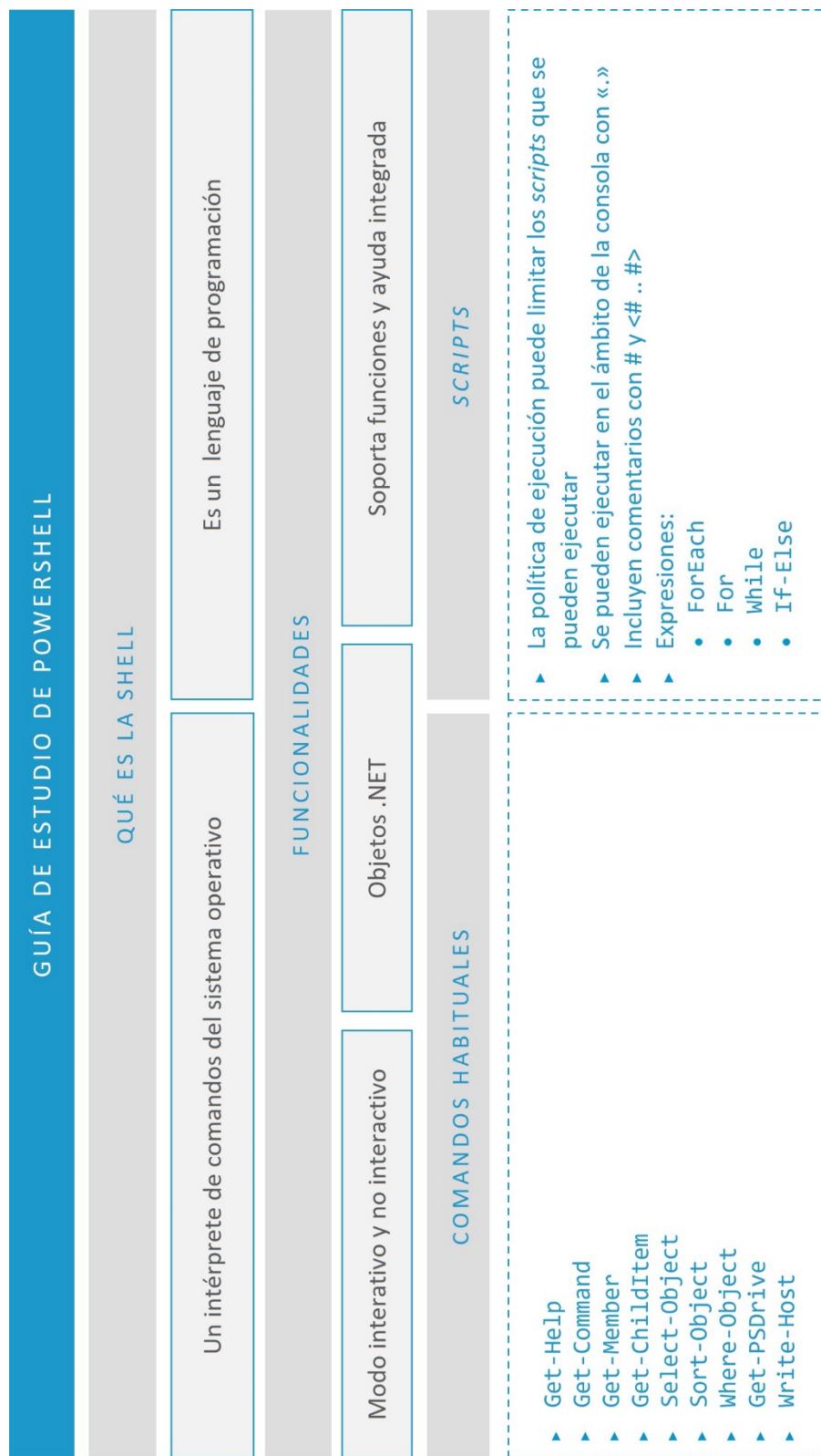
Administración de Sistemas en la Cloud

Guía de estudio de PowerShell

Índice

Esquema	3
Ideas clave	4
7.1. Introducción y objetivos	4
7.2. Qué es PowerShell	4
7.3. <i>Cmdlets</i>	5
7.4. <i>Scripts</i>	11
7.5. Expresiones y otros comandos	20
7.6. Referencias bibliográficas	24

Esquema



7.1. Introducción y objetivos

Este tema es una guía de las funcionalidades básicas de PowerShell. Explicará el concepto de *cmdlet* y qué características comunes tienen. También se explicarán los *cmdlets* básicos y cómo combinarlos para escribir *scripts*. Para finalizar, se mencionarán algunas de las expresiones de programación presentes en PowerShell.

Los **objetivos** que se pretenden conseguir son:

- ▶ Conocer los fundamentos de los conceptos básicos de PowerShell.
- ▶ Aprender a usar los *cmdlets* básicos.
- ▶ Aprender a combinar los *cmdlets* con expresiones del lenguaje.

7.2. Qué es PowerShell

PowerShell es una **consola de línea de comandos de Windows** diseñada especialmente para administradores. Incluye una *shell* interactiva y un entorno de *scripting* al estilo de consolas para Linux como Bash. El entorno interactivo y los *scripts* pueden usarse de manera independiente o en conjunto (Shepard, 2015).

A diferencia de otras *shells*, cuyos comandos devuelven y aceptan texto, [PowerShell](#) está basada en el *framework* .NET y, como tal, devuelve y acepta objetos de .NET. Este cambio de paradigma ofrece funcionalidades muy útiles para tareas administrativas.

PowerShell introduce el concepto de *cmdlet*, o *commandlet*: un comando propio de la *shell*, al estilo de los comandos *built-in* de Bash. Los *cmdlets* pueden operar en conjunto, de manera que la salida de un comando puede alimentar la entrada de otro, siguiendo el estilo de las tuberías de otras consolas.

PowerShell permite a los administradores acceder al sistema de ficheros, al igual que otras consolas, además de recursos específicos de Windows, como el registro y los almacenes de certificados de seguridad.

Desde su introducción en 2007, PowerShell es el entorno de *scripting* por defecto de la mayoría de *sysadmins*. Es posible automatizar prácticamente cualquier tarea y, cuando no hay un *cmdlet* para una cierta tarea, siempre es posible invocar comandos nativos de Windows.

7.3. *Cmdlets*

Los comandos de otras consolas tienen nombres y parámetros definidos de manera irregular. Sin embargo, los *cmdlets* de PowerShell siguen un esquema común de **verbo-nombre**. Los verbos no tienen por qué ser un verbo gramaticalmente hablando, sino que expresan la acción que ejecutará el comando sobre un recurso. El recurso, expresado por el segundo componente, describe objetos específicos del sistema operativo, el sistema de ficheros, etc. Por ejemplo, algunos de los *cmdlets* básicos son Get-Process, Stop-Process, Get-Service y Stop-Service.

A continuación, en el vídeo *Primeros pasos en PowerShell*, se muestran algunos de los comandos básicos de PowerShell y una explicación básica de cómo guardar y ejecutar *scripts*.



Accede al vídeo

El concepto de verbo y nombre no se limita al texto con el que invocar el comando. El comando `Get-Command`, muy utilizado para obtener ayuda de forma interactiva, lista todos los comandos disponibles en un entorno concreto. Uno de los filtros que acepta es el nombre sobre el que operan los comandos. Es posible obtener la lista de comandos que operan sobre un **recurso concreto**, `Process`, por ejemplo, de forma inmediata:

```
PS C:\> Get-Command -Noun process
```

CommandType	Name	Version	Source
Cmdlet	Debug-Process	3.1.0.0	Microsoft.PowerShell.Management
Cmdlet	Get-Process	3.1.0.0	Microsoft.PowerShell.Management
Cmdlet	Start-Process	3.1.0.0	Microsoft.PowerShell.Management
Cmdlet	Stop-Process	3.1.0.0	Microsoft.PowerShell.Management
Cmdlet	Wait-Process	3.1.0.0	Microsoft.PowerShell.Management

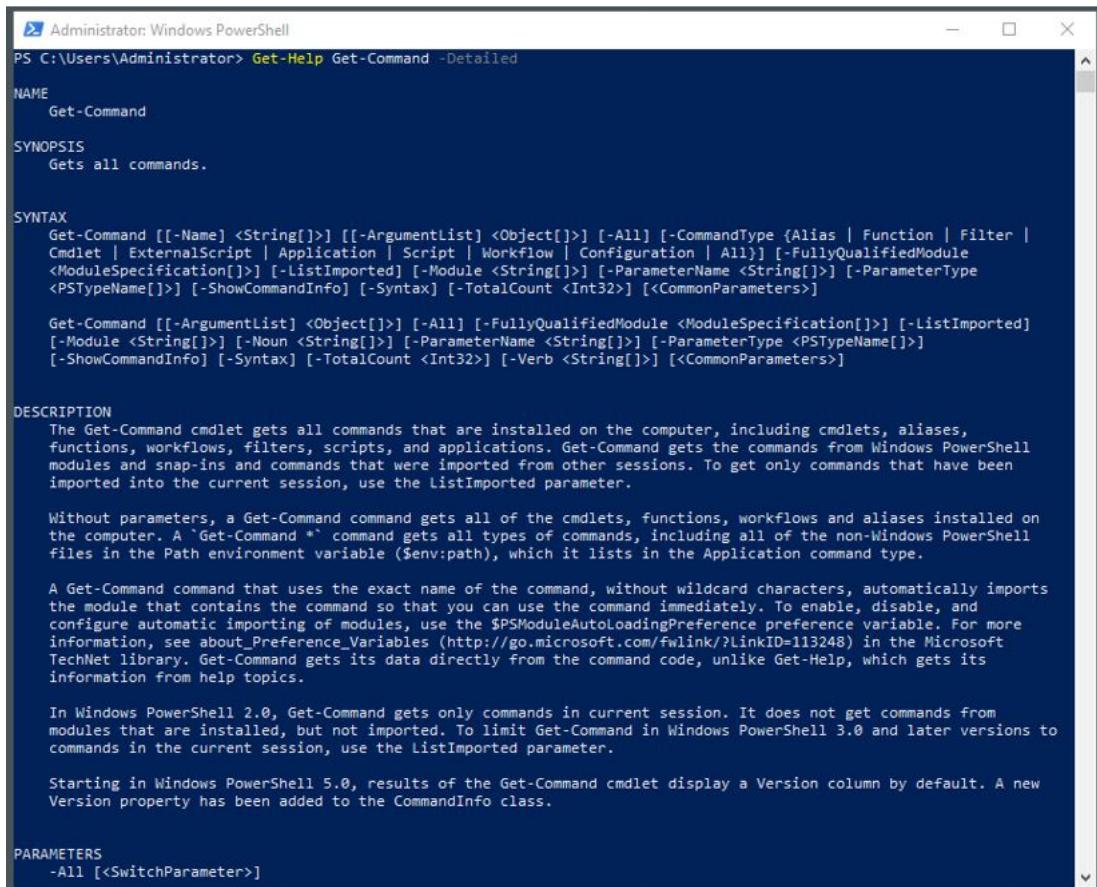
Al contrario que otras interfaces de línea de comandos, PowerShell **procesa los parámetros directamente**, por lo que la estructura de estos es homogénea entre comandos diferentes. Los comandos estándar de la librería siguen una nomenclatura homogénea, lo que facilita su uso. Además, todos los comandos incluyen una serie de parámetros comunes controlados por el motor de PowerShell: `WhatIf`, `Confirm`, `Verbose`, `Debug`, `Warn`, `ErrorAction`, `ErrorVariable`, `OutVariable`, y `OutBuffer`.

Comandos básicos

El primer comando básico ya se ha mencionado: `Get-Command`. Es fundamental para averiguar los comandos disponibles en un equipo. PowerShell es extensible y es habitual que las herramientas de administración incorporen módulos de PowerShell adicionales. Por tanto, la salida de `Get-Command` variará entre un sistema y otro.

Otro comando ideal para obtener información es `Get-Help` (Hill, 2009). Este *cmdlet* proporciona ayuda sobre lo que hace un *cmdlet* específico, sus parámetros e incluso incluye ejemplos con varios casos de uso. Por ejemplo, `Get-Help Get-Command -`

Detailed mostrará una descripción del comando y un listado de los parámetros con el tipo de dato que espera cada uno, tal como muestra la Figura 1.



The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell" with the command "Get-Help Get-Command -Detailed" entered. The output is as follows:

```
PS C:\Users\Administrator> Get-Help Get-Command -Detailed

NAME
  Get-Command

SYNOPSIS
  Gets all commands.

SYNTAX
  Get-Command [[-Name] <String[]>] [[-ArgumentList] <Object[]>] [-All] [- CommandType {Alias | Function | Filter | Cmdlet | ExternalScript | Application | Script | Workflow | Configuration | All}] [-FullyQualifiedModule <ModuleSpecification[]>] [-ListImported] [-Module <String[]>] [-ParameterName <String[]>] [-ParameterType <PSTypeName[]>] [-ShowCommandInfo] [-Syntax] [-TotalCount <Int32>] [<CommonParameters>]

  Get-Command [[-ArgumentList] <Object[]>] [-All] [-FullyQualifiedModule <ModuleSpecification[]>] [-ListImported] [-Module <String[]>] [-Noun <String[]>] [-ParameterName <String[]>] [-ParameterType <PSTypeName[]>] [-ShowCommandInfo] [-Syntax] [-TotalCount <Int32>] [-Verb <String[]>] [<CommonParameters>]

DESCRIPTION
  The Get-Command cmdlet gets all commands that are installed on the computer, including cmdlets, aliases, functions, workflows, filters, scripts, and applications. Get-Command gets the commands from Windows PowerShell modules and snap-ins and commands that were imported from other sessions. To get only commands that have been imported into the current session, use the ListImported parameter.

  Without parameters, a Get-Command command gets all of the cmdlets, functions, workflows and aliases installed on the computer. A `Get-Command *` command gets all types of commands, including all of the non-Windows PowerShell files in the Path environment variable ($env:path), which it lists in the Application command type.

  A Get-Command command that uses the exact name of the command, without wildcard characters, automatically imports the module that contains the command so that you can use the command immediately. To enable, disable, and configure automatic importing of modules, use the $PSModuleAutoLoadingPreference preference variable. For more information, see about_Preference_Variables (http://go.microsoft.com/fwlink/?LinkId=113248) in the Microsoft TechNet library. Get-Command gets its data directly from the command code, unlike Get-Help, which gets its information from help topics.

  In Windows PowerShell 2.0, Get-Command gets only commands in current session. It does not get commands from modules that are installed, but not imported. To limit Get-Command in Windows PowerShell 3.0 and later versions to commands in the current session, use the ListImported parameter.

  Starting in Windows PowerShell 5.0, results of the Get-Command cmdlet display a Version column by default. A new Version property has been added to the CommandInfo class.

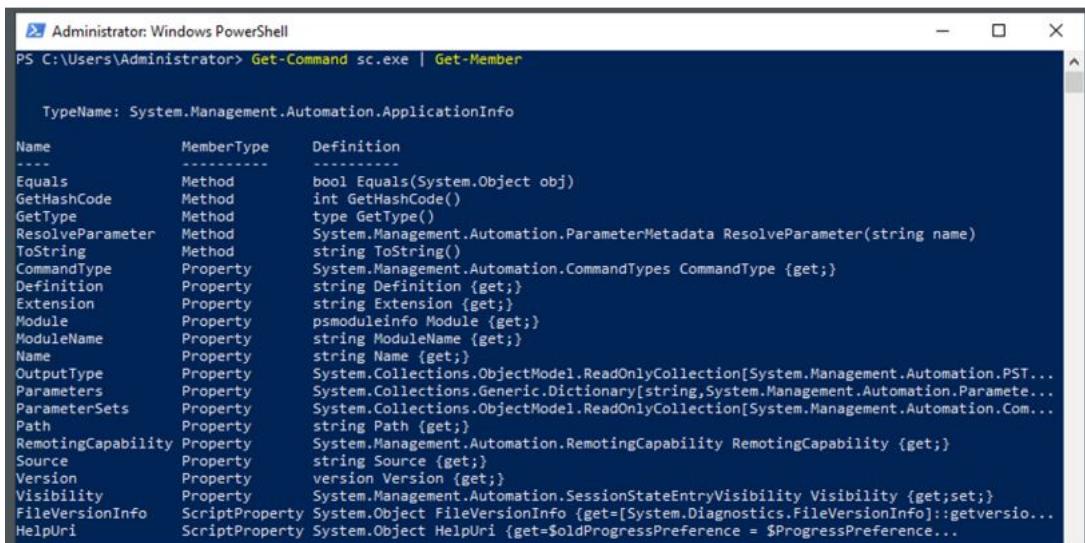
PARAMETERS
  -All [<SwitchParameter>]
```

Figura 1. Comando Get-Help Get-Command. Fuente: elaboración propia.

Windows Server 2019 no incluye todos los ficheros de ayuda por defecto. Get-Help puede redirigir a la web de Microsoft para obtener ayuda específica, pero también es posible descargar todos los ficheros con Update-Help.

Otro de los comandos más utilizados es **Get-Member**. Uno de los conceptos que más cuesta asumir es que prácticamente todo lo que manipulan los *cmdlets* son objetos .NET. Esto significa que, al enviar la salida en una tubería de un comando a otro, el segundo comando recibirá un objeto. El tipo de los objetos o su formato no tienen por qué ser siempre conocidos, y aquí es donde Get-Member aporta su valor: mostrará el detalle de los métodos y atributos del objeto que haya recibido. Por ejemplo, en la Figura 2, Get-Member recibe un objeto de tipo ApplicationInfo. Este objeto tiene

algunos métodos estándar, heredados de una clase padre, y tiene una lista de propiedades, la mayoría de tipo *string*.



```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Get-Command sc.exe | Get-Member

TypeName: System.Management.Automation.ApplicationInfo

Name      MemberType      Definition
----      -----      -----
Equals    Method      bool Equals(System.Object obj)
GetHashCode Method      int GetHashCode()
GetType   Method      type GetType()
ResolveParameter Method      System.Management.Automation.ParameterMetadata ResolveParameter(string name)
ToString   Method      string ToString()
CommandType Property    System.Management.Automation.CommandType CommandType {get;}
Definition Property    string Definition {get;}
Extension Property    string Extension {get;}
Module    Property    psmoduleinfo Module {get;}
ModuleName Property    string moduleName {get;}
Name      Property    string Name {get;}
OutputType Property    System.Collections.ObjectModel.ReadOnlyCollection[System.Management.Automation.PST...
Parameters Property    System.Collections.Generic.Dictionary[string, System.Management.Automation.Paramete...
ParameterSets Property    System.Collections.ObjectModel.ReadOnlyCollection[System.Management.Automation.Com...
Path      Property    string Path {get;}
RemotingCapability Property    System.Management.Automation.RemotingCapability RemotingCapability {get;}
Source    Property    string Source {get;}
Version   Property    version Version {get;}
Visibility Property    System.Management.Automation.SessionStateEntryVisibility Visibility {get;set;}
FileVersionInfo FileVersionInfo System.Object FileVersionInfo {get=[System.Diagnostics.FileVersionInfo]:getversio...
HelpUri   ScriptProperty System.Object HelpUri {get=$oldProgressPreference = $ProgressPreference...}
```

Figura 2. Ejemplo de Get-Member. Fuente: elaboración propia.

Visto que el objeto tiene una propiedad *Version*, es posible acceder a ella igual que en otros lenguajes de programación: con el operador «.» (punto), como muestra la Figura 3.



```
PS C:\Users\Administrator> (Get-Command sc.exe).Version

Major Minor Build Revision
---- ---- 17763 1
```

Figura 3. Acceso a propiedades de un objeto de PowerShell. Fuente: elaboración propia.

Si la salida del comando produce una lista de objetos, Get-Member dará información sobre los objetos como tal. Con esta información, es posible acceder al contenido de la lista con otras expresiones nativas de PowerShell, como *foreach*. Esta expresión aplica un bucle sobre una lista. En cada iteración, la variable *\$_.* contiene el objeto correspondiente y es posible acceder a él, como en el caso anterior. La Figura 4 muestra un ejemplo en el que la lista contiene objetos de tipo *CmdletInfo*.

```
PS C:\> Get-Command -Noun Process | foreach {$_ .Name + " " + $_ .Version}
Debug-Process 3.1.0.0
Get-Process 3.1.0.0
Start-Process 3.1.0.0
Stop-Process 3.1.0.0
Wait-Process 3.1.0.0
PS C:\>
```

Figura 4. Get-Command y foreach. Fuente: elaboración propia.

El último comando básico de esta sección es **Get-PSDrive**. En PowerShell, el sistema de ficheros es solo uno de los tipos de unidades que se pueden manipular. Las unidades disponibles se pueden obtener con Get-PSDrive, sin parámetros (ver Figura 5).

```
PS C:\> Get-PSDrive
Name      Used (GB)   Free (GB) Provider    Root
----      -----     -----   -----
Alias
C          10.78      20.69  FileSystem   C:\
Cert
D          0.05       0.00   FileSystem   D:\
Env
Function
HKCU
HKLM
Variable
WSMan
```

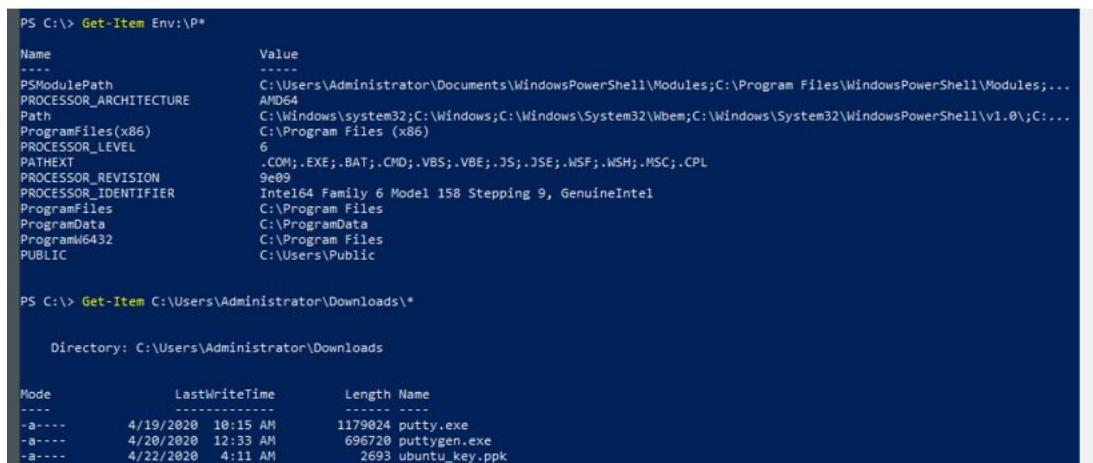
Figura 5. Lista de unidades con Get-PSDrive. Fuente: elaboración propia.

Estas unidades pueden manipularse con el mismo subconjunto de *cmdlets*, que se pueden obtener con Get-Command *-Item* (ver Figura 6).

```
PS C:\> Get-Command *-Item*
 CommandType   Name
-----   -----
 Cmdlet      Clear-Item
 Cmdlet      Clear-ItemProperty
 Cmdlet      Copy-Item
 Cmdlet      Copy-ItemProperty
 Cmdlet      Get-Item
 Cmdlet      Get-ItemProperty
 Cmdlet      Get-ItemPropertyValue
 Cmdlet      Invoke-Item
 Cmdlet      Move-Item
 Cmdlet      Move-ItemProperty
 Cmdlet      New-Item
 Cmdlet      New-ItemProperty
 Cmdlet      Remove-Item
 Cmdlet      Remove-ItemProperty
 Cmdlet      Rename-Item
 Cmdlet      Rename-ItemProperty
 Cmdlet      Set-Item
 Cmdlet      Set-ItemProperty
```

Figura 6. Comandos para manipulación de unidades. Fuente: elaboración propia.

Por ejemplo, la Figura 7 demuestra el uso de `Get-Item` para obtener un listado de los archivos de `c:\Users\Administrator\Downloads` y para obtener las variables de entorno que empiezan con P.



```
PS C:\> Get-Item Env:\P*
Name          Value
----          -----
PSModulePath  C:\Users\Administrator\Documents\WindowsPowerShell\Modules;C:\Program Files\WindowsPowerShell\Modules;...
PROCESSOR_ARCHITECTURE  AMD64
Path          C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:...
ProgramFiles(x86)  C:\Program Files (x86)
PROCESSOR_LEVEL  6
PATHEXT       .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.MSF;.WSH;.MSC;.CPL
PROCESSOR_REVISION  9e09
PROCESSOR_IDENTIFIER  Intel64 Family 6 Model 158 Stepping 9, GenuineIntel
ProgramFiles   C:\Program Files
ProgramData    C:\ProgramData
ProgramW6432   C:\Program Files
PUBLIC        C:\Users\Public

PS C:\> Get-Item C:\Users\Administrator\Downloads\*

Directory: C:\Users\Administrator\Downloads

Mode          LastWriteTime      Length Name
----          -----          Length Name
-a---  4/19/2020 10:15 AM     1179024 putty.exe
-a---  4/20/2020 12:33 AM      696720 puttygen.exe
-a---  4/22/2020  4:11 AM        2693 ubuntu_key.ppk
```

Figura 7. `Get-Item` en unidad de sistema de archivos y en variables de entorno. Fuente: elaboración propia.

Alias

PowerShell soporta alias, que son nombres alternativos de comandos y *cmdlets* (Shepard, 2015). Por ejemplo, `dir` y `ls` son alias de `Get-ChildItem` y `cd` es un alias de `Set-Location`. Hay alias disponibles para muchos *cmdlets* con funcionalidad similar en entornos DOS o Linux. El objetivo es doble: hacer el código más conciso en la línea de comandos y facilitar la transición de los usuarios habituales de otras *shell*.

Para obtener una lista de los alias definidos en una sesión, se puede hacer uso de `Get-Alias`. Con el parámetro por defecto, `Get-Alias <alias>`, se obtiene el comando al que un alias hace referencia y con `Get-Alias -Definition <cmdlet>` se obtienen los alias definidos para ese *cmdlet* (ver Figura 8).

```

PS C:\> Get-Alias ls
CommandType      Name
----          ----
Alias           ls -> Get-ChildItem

PS C:\> Get-Alias -Definition Get-ChildItem
CommandType      Name
----          ----
Alias           dir -> Get-ChildItem
Alias           gci -> Get-ChildItem
Alias           ls -> Get-ChildItem

```

Figura 8. Alias de Child-Item. Fuente: elaboración propia.

El cmdlet New-Alias crea un alias en la sesión actual. Los alias no son permanentes y desaparecen al cerrar la consola, pero es posible guardarlos con Export-Alias. Al iniciar sesión, se pueden importar con Import-Alias.

```

PS C:\> New-Alias -Name ll -Value Get-ChildItem
PS C:\> ll C:\Users\Administrator\Downloads\

Directory: C:\Users\Administrator\Downloads

Mode          LastWriteTime     Length Name
----          -----          ----
-a---  4/19/2020 10:15 AM    1179024 putty.exe
-a---  4/20/2020 12:33 AM    696720 puttygen.exe
-a---  4/22/2020 4:11 AM      2693 ubuntu_key.ppk

PS C:\> Export-Alias -Path C:\Users\Administrator\alias.csv
PS C:\> Import-Alias -Path C:\Users\Administrator\alias.csv -ErrorAction SilentlyContinue

```

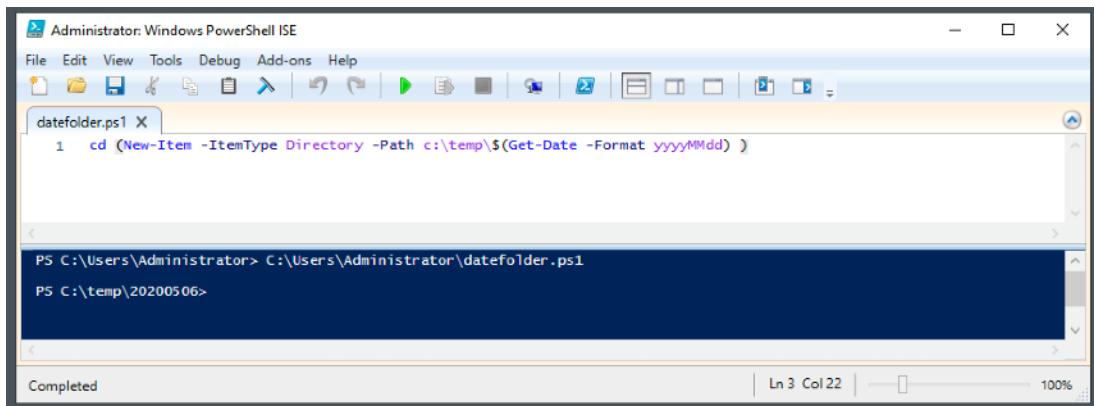
Figura 9. Gestión de alias. Fuente: elaboración propia.

7.4. Scripts

Como se ha mencionado anteriormente, PowerShell ofrece no solo una herramienta interactiva, sino también un **entorno de programación**. Los *scripts* en PowerShell tienen la extensión .ps1, independientemente de la versión de PowerShell (Shepard, 2015).

Aunque, independiente del motor de ejecución, la mayoría de las versiones de Windows incluyen un editor llamado PowerShell ISE. Ofrece autocompletado de los nombres de los comandos y de los parámetros, igual que la línea de comandos, así

como ejecución del *script*, *debugging*, coloreado de sintaxis, etc. La Figura 10 muestra el PowerShell ISE con un *script* sencillo que crea una carpeta nueva y cambia a ella.



The screenshot shows the Windows PowerShell ISE interface. The title bar says "Administrator: Windows PowerShell ISE". The menu bar includes File, Edit, View, Tools, Debug, Add-ons, and Help. The toolbar has various icons for file operations. A tab at the top is labeled "datefolder.ps1 X". The main code editor area contains the following PowerShell command:

```
1 cd (New-Item -ItemType Directory -Path c:\temp\$((Get-Date -Format yyyyMMdd)) )
```

Below the code editor is a dark blue command window. It shows the command being run and the resulting directory path:

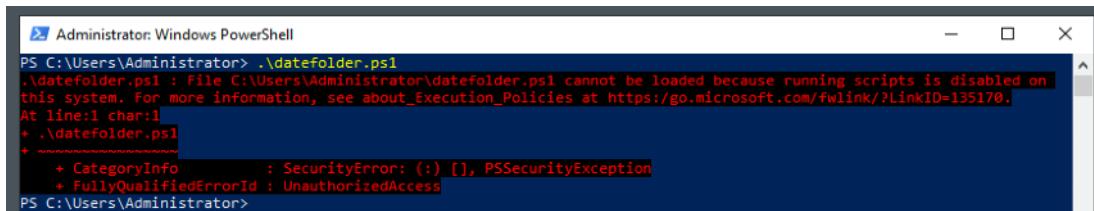
```
PS C:\Users\Administrator> C:\Users\Administrator\datefolder.ps1
PS C:\temp\20200506>
```

The status bar at the bottom right indicates "Completed".

Figura 10. PowerShell ISE. Fuente: elaboración propia.

Política de ejecución

El *script* anterior se ha ejecutado en el propio editor, pero también es posible ejecutarlo en la línea de comandos. Sin embargo, es habitual que ocurra algo parecido a la Figura 11, donde se muestra un mensaje de error que hace referencia a la política de ejecución o *execution policy*.



The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command being run is ".\datefolder.ps1". The output shows an error message related to execution policies:

```
PS C:\Users\Administrator> .\datefolder.ps1
.\datefolder.ps1 : File C:\Users\Administrator\dateFolder.ps1 cannot be loaded because running scripts is disabled on
this system. For more information, see about_Execution_Policies at https://go.microsoft.com/fwlink/?LinkID=135170.
At line:1 char:1
+ .\dateFolder.ps1
+ ~~~~~
+ CategoryInfo          : SecurityError: () [], PSSecurityException
+ FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\Administrator>
```

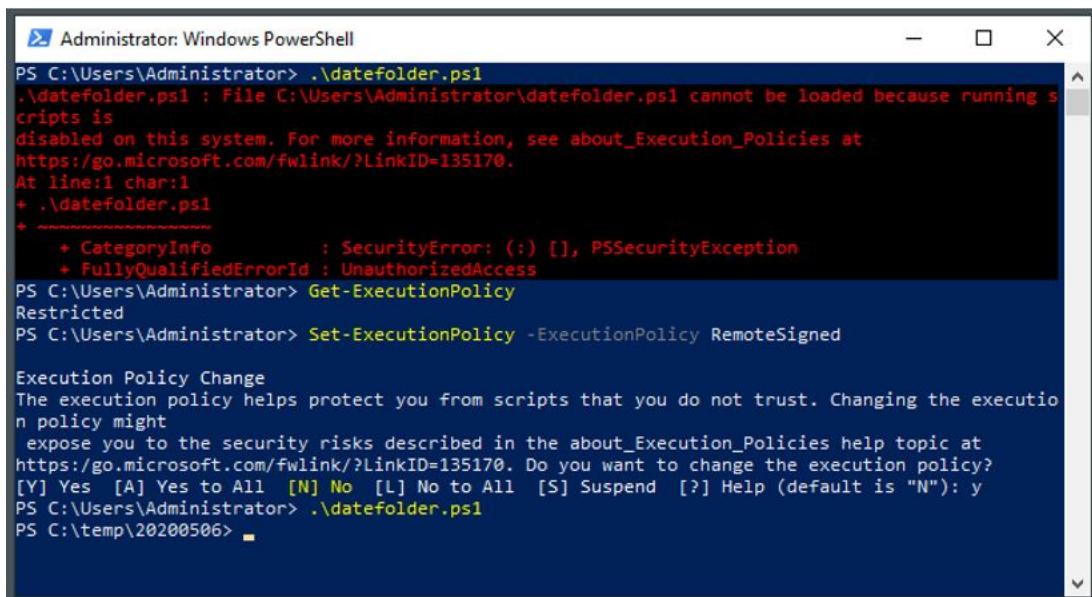
Figura 11. Error por restricción de la política de ejecución. Fuente: elaboración propia.

La política de ejecución es una **medida de seguridad** de PowerShell que ofrece control a los administradores sobre qué *scripts* se pueden ejecutar. Las políticas de ejecución posibles son:

- ▶ *Restricted*: no permite la ejecución de *scripts* en ningún caso. Fue la política por defecto para las versiones anteriores a Windows Server 2012 R2.
- ▶ *All signed*: solo los *scripts* con una firma digital válida se pueden ejecutar.

- ▶ *Remote signed*: los *scripts* de ubicaciones remotas deben tener una firma digital válida para ejecutarlos, pero los *scripts* locales se pueden ejecutar sin restricciones. Es la política por defecto desde Windows Server 2012 R2.
- ▶ *Unrestricted*: cualquier *script* se puede ejecutar sin restricciones.

El comando `Get-ExecutionPolicy` muestra la política actual, mientras que `Set-ExecutionPolicy` permite cambiar de una política a otra (ver Figura 12).



The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command `.\datefolder.ps1` is run, resulting in an error message about execution policies. Then, `Get-ExecutionPolicy` shows the current policy is "Restricted". Finally, `Set-ExecutionPolicy -ExecutionPolicy RemoteSigned` is run, changing the policy. A confirmation message asks if the user wants to change the execution policy, with options [Y] Yes, [A] Yes to All, [N] No, [L] No to All, [S] Suspend, and [?] Help. The user selects [Y]. The command `.\datefolder.ps1` is then run again successfully.

```

Administrator: Windows PowerShell
PS C:\Users\Administrator> .\datefolder.ps1
.\datefolder.ps1 : File C:\Users\Administrator\datefolder.ps1 cannot be loaded because running scripts is
disabled on this system. For more information, see about_Execution_Policies at
https://go.microsoft.com/fwlink/?LinkId=135170.
At line:1 char:1
+ .\datefolder.ps1
+ ~~~~~
    + CategoryInfo          : SecurityError: () [], PSSecurityException
    + FullyQualifiedErrorId : UnauthorizedAccess
PS C:\Users\Administrator> Get-ExecutionPolicy
Restricted
PS C:\Users\Administrator> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned
Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might
expose you to the security risks described in the about_Execution_Policies help topic at
https://go.microsoft.com/fwlink/?LinkId=135170. Do you want to change the execution policy?
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "N"): y
PS C:\Users\Administrator> .\datefolder.ps1
PS C:\temp\20200506>

```

Figura 12. Cambio de política de ejecución. Fuente: elaboración propia.

Tipos de *scripts*

Una distinción habitual divide los *scripts* en controladores y herramientas. Los **scripts controladores** llaman a otros *scripts*, *cmdlets* y funciones para completar su tarea. Pueden incluir más de una tarea, pero en general no se espera que se vayan a reusar a menudo. Se usan habitualmente, por ejemplo, en el Programador de tareas de Windows.

Por otro lado, los **scripts herramientas** ejecutan una única tarea y están pensados, precisamente, para ser reutilizados. Se usarían individualmente, únicamente en una sesión interactiva, pero el objetivo es realmente ofrecer una solución a un paso

concreto dentro de una tarea más amplia. No hay diferencias técnicas entre ambos tipos, sino más bien un estilo diferente de ejecución y documentación elegido por el administrador que lo escribió.

Comentarios

Los *scripts* de PowerShell aceptan comentarios de una línea precedidos por el carácter # y de varias líneas rodeados con los delimitadores <# y #>.

```
# comentario en una linea  
Write-Host Hello World
```

```
<#  
    comentario en  
    multiples lineas  
#>  
Write-Host Bye bye
```

Ámbito de los *scripts*

El motor de PowerShell crea un **ámbito** (o *scope*) durante la ejecución de cada *script*. Si el *script* crea o modifica variables o funciones, estos cambios son visibles en el ámbito del *script* y desaparecerán cuando el *script* termine. En este ejemplo, la variable \$var se crea en el ámbito del *script* hello-world.ps1, por lo que al terminar el *script* ya no está disponible:

```
PS C:\> Get-Content .\hello-world.ps1  
$var = "Hello World"  
Write-Host "La variable contiene:" $var  
PS C:\> .\hello-world.ps1  
La variable contiene: Hello World  
PS C:\> Write-Host $var  
  
PS C:\>
```

PowerShell usa la misma técnica que Bash para ejecutar un *script* en el ámbito de la *shell* actual para mantener todos los objetos creados durante la ejecución: en vez invocar el *script* como cualquier otro ejecutable, se invoca el *script* con el comando «.» (punto). En Bash, el punto era equivalente al comando `source`, esta herramienta suele llamarse *dot-sourcing*. Este método tiene el mismo efecto que ejecutar todos los comandos del *script* en la línea de comandos, uno tras otro. El resultado del ejemplo anterior, usando esta técnica, sería el siguiente:

```
PS C:\> . .\hello-world.ps1
La variable contiene: Hello World
PS C:\> echo $var
Hello World
PS C:\>
```

Parámetros

Ejecutar una serie de comandos puede ser útil, pero es habitual que estos comandos necesiten algún tipo de datos de entrada. PowerShell soporta la definición de múltiples parámetros, el tipo, los valores por defecto y la obligatoriedad. Esta funcionalidad es nativa de PowerShell, por lo que no hay que usar comandos adicionales, como `getopts` en Bash, para pasear los parámetros.

PowerShell incluye este tipo de validaciones, pero eso no implica que el *script* pueda incluir validaciones adicionales específicas para la lógica del código. Por ejemplo, un parámetro *puerto* puede estar definido como un entero, que PowerShell puede comprobar, pero haría falta una comprobación adicional para verificar que el valor entra dentro de un rango concreto de enteros. El siguiente código muestra un ejemplo de esta funcionalidad:

```
Param(
[Parameter(Mandatory=$true)]
[string] $Var1,
[Parameter(Mandatory=$false)]
[Int32] $Var2=42
```

```

)
Write-Host "Var1 contiene:" $Var1
Write-Host "Var2 contiene:" $Var2

```

El parámetro Var1 es obligatorio y está definido como una cadena de texto, mientras que Var2 será un entero con valor 42, si el usuario no especifica otro valor. La Figura 13 muestra varias ejecuciones del *script* en las que PowerShell:

- ▶ Sigue la ejecución del script.
- ▶ Pide al usuario que proporcione el valor para el parámetro.
- ▶ Muestra el resultado de la ejecución.
- ▶ Muestra un error si el valor de un parámetro no es conforme al tipo especificado.

```

Administrator: Windows PowerShell
PS C:\Users\Administrator> .\hello-world.ps1
cmdlet hello-world.ps1 at command pipeline position 1
Supply values for the following parameters:
Var1: Hello World
Var1 contiene: Hello World
Var2 contiene: 42
PS C:\Users\Administrator> .\hello-world.ps1 -Var1 "Hola Mundo" -Var2 53
Var1 contiene: Hola Mundo
Var2 contiene: 53
PS C:\Users\Administrator> .\hello-world.ps1 -Var1 "Hola Mundo" -Var2 Hello
C:\Users\Administrator\hello-world.ps1 : Cannot process argument
transformation on parameter 'Var2'. Cannot convert value "Hello" to type
"System.Int32". Error: "Input string was not in a correct format."
At line:1 char:44
+ .\hello-world.ps1 -Var1 "Hola Mundo" -Var2 Hello
+
+     + CategoryInfo          : InvalidData: (:) [hello-world.ps1], ParameterB
indingArgumentTransformationException
+ FullyQualifiedErrorId : ParameterArgumentTransformationError,hello-wor
ld.ps1
PS C:\Users\Administrator>

```

Figura 13. Validación de parámetros en un *script*. Fuente: elaboración propia.

Funciones

Al igual que otros *frameworks* de programación y *scripting*, PowerShell soporta la definición de funciones. La definición de funciones y *scripts* es similar: cualquier línea de código que se pueda escribir en un *script* se puede escribir en una función y las

funciones aceptan parámetros al igual que los *scripts*. No obstante, las funciones permiten reusar código y facilitan su mantenimiento, como en cualquier lenguaje de programación. Además, PowerShell lista todas las funciones disponibles en una de las unidades, Function:, tal como muestra la Figura 14.

CommandType	Name	Version
Function	A:	
Function	B:	
Function	C:	
Function	cd..	
Function	cd\	
Function	Clear-Host	
Function	ConvertFrom-SddlString	3.1.0.0
Function	D:	
Function	E:	
Function	F:	

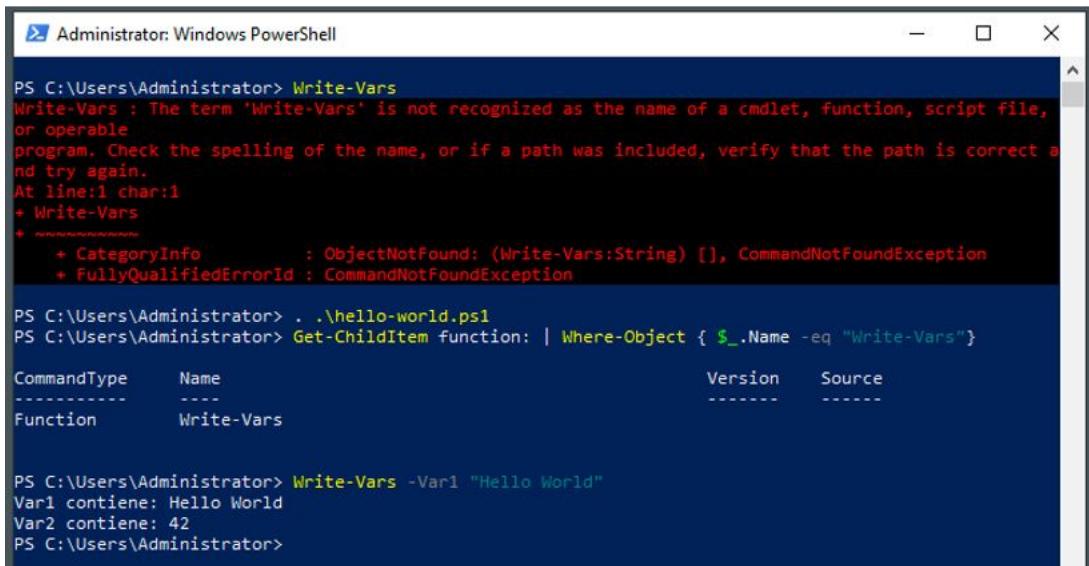
Figura 14. Lista de funciones. Fuente: elaboración propia.

La definición de una función se limita a usar la palabra function, englobando el código entre llaves. La definición de parámetros es idéntica a la de un *script* y debe estar al principio de la función. El ejemplo de *script* de la sección anterior se puede convertir en una función de la siguiente manera.

```
function Write-Vars {  
    Param(  
        [Parameter(Mandatory=$true)]  
        [string] $Var1,  
        [Parameter(Mandatory=$false)]  
        [Int32] $Var2=42  
    )  
  
    Write-Host "Var1 contiene:" $Var1  
    Write-Host "Var2 contiene:" $Var2  
}
```

Al ejecutar este *script*, que solo contiene una función, no se imprimirá nada en la consola. El *script* ya no ejecuta los cmdlets de cada línea, sino que define una función, que estará disponible en el resto del ámbito del *script*. Sin embargo, si el *script* se

ejecuta con un *dot-source*, la función estará disponible en el ámbito de la consola y se podrá ejecutar como un comando más (ver Figura 15).



```
Administrator: Windows PowerShell
PS C:\Users\Administrator> Write-Vars
Write-Vars : The term 'Write-Vars' is not recognized as the name of a cmdlet, function, script file,
or operable
program. Check the spelling of the name, or if a path was included, verify that the path is correct a
nd try again.
At line:1 char:1
+ Write-Vars
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (Write-Vars:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

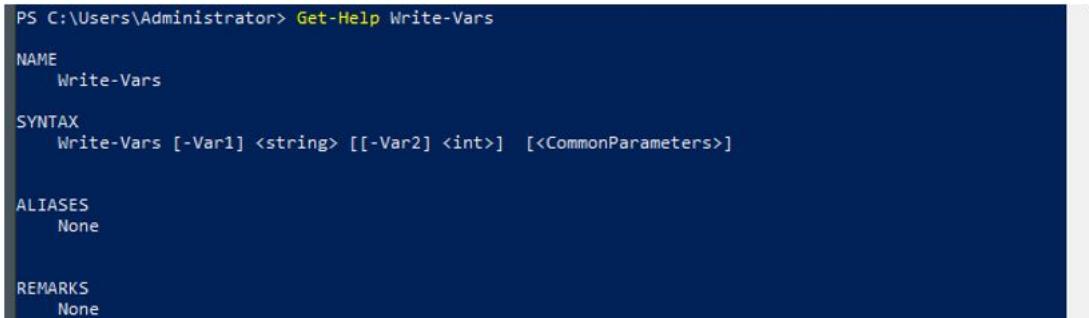
PS C:\Users\Administrator> .\hello-world.ps1
PS C:\Users\Administrator> Get-ChildItem function: | Where-Object { $_.Name -eq "Write-Vars" }

 CommandType      Name          Version      Source
 -----          ----          -----      -----
 Function        Write-Vars

PS C:\Users\Administrator> Write-Vars -Var1 "Hello World"
Var1 contiene: Hello World
Var2 contiene: 42
PS C:\Users\Administrator>
```

Figura 15. Definición de función en un *script*. Fuente: elaboración propia.

La simple definición de una función incorpora una ayuda básica, accesible con Get-Help (ver Figura 16).



```
PS C:\Users\Administrator> Get-Help Write-Vars

NAME
    Write-Vars

SYNTAX
    Write-Vars [-Var1] <string> [[-Var2] <int>]  [<CommonParameters>]

ALIASES
    None

REMARKS
    None
```

Figura 16. Ayuda básica de una función personalizada. Fuente: elaboración propia.

PowerShell soporta unos comentarios con un formato especial, para personalizar la ayuda. Las palabras clave como .Synopsis y .DESCRIPTION delimitan cada una de las secciones de la ayuda. La función del ejemplo anterior se podría comentar de la siguiente manera:

```

<#
.Synopsis
    Funcion de ejemplo
.DESCRIPTION
    Esta funcion imprime los valores de dos parametros, Var1,
    que es una cadena de texto y es obligatorio, y Var2, que
    es un entero y su valor por defecto es 42.
.EXAMPLE
    Write-Vars -Var1 "Hello World"
.EXAMPLE
    Write-Vars -Var1 "Hello World" -Var2 53
.INPUTS
    No espera datos por la entrada estandar.
.OUTPUTS
    Emite el parametro Var2 por la salida estandar.

#>
function Write-Vars {
    Param(
        [Parameter(Mandatory=$true)]
        [string] $Var1,
        [Parameter(Mandatory=$false)]
        [Int32] $Var2=42
    )

    Write-Host "Var1 contiene:" $Var1
    Write-Host "Var2 contiene:" $Var2
    $Var2
}

```

Una vez importada en la consola, la ayuda contendrá las secciones definidas en el comentario (ver Figura 17).

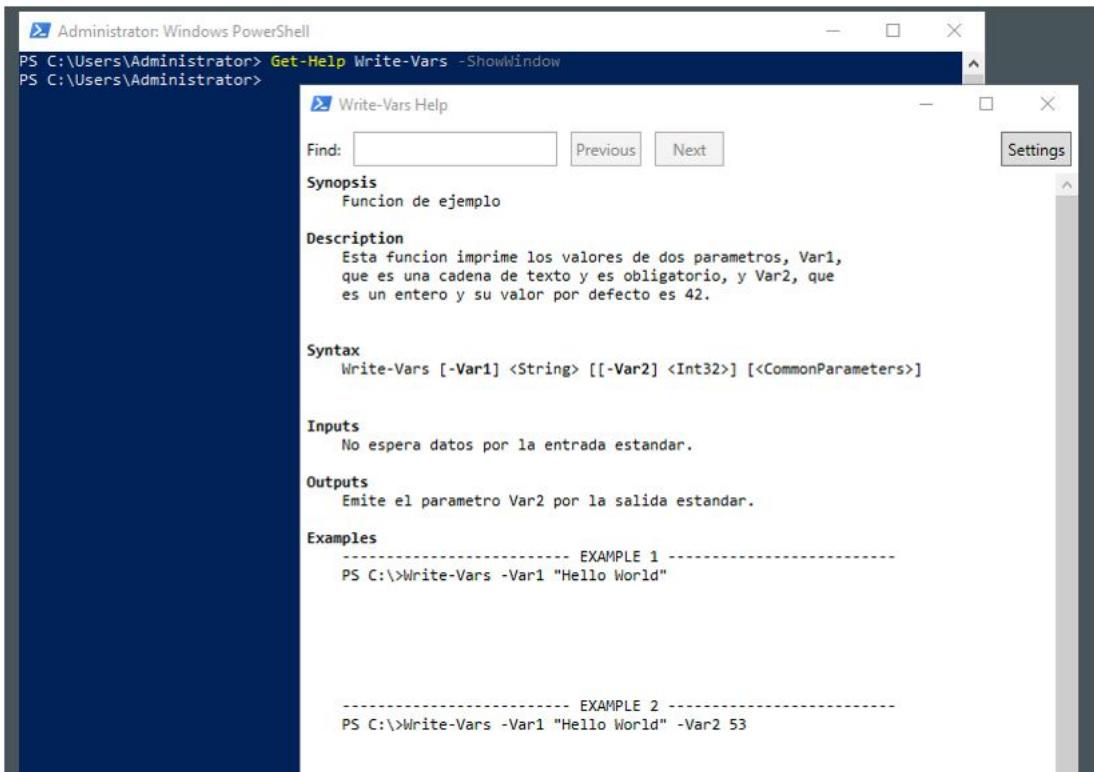


Figura 17. Ayuda personalizada. Fuente: elaboración propia.

7.5. Expresiones y otros comandos

If-Else

Los bloques lógicos *If-Else* funcionan de manera esperada, como en otros lenguajes de programación:

```
If ($Var1 -eq $Var2) {  
    Write-Host "Iguales"  
} ElseIf ($Var1 -gt $Var2) {  
    Write-Host "Mayor"  
} Else {  
    Write-Host "Menor"  
}
```

Los operadores de igualdad, mayor que, etc., se parecen más a los usados en Bash que a los disponibles en otros lenguajes. La Tabla 1 contiene algunos de los más habituales. Se puede consultar la tabla completa [aquí](#).

Comparadores lógicos		
	Significado	Ejemplo
-ne	<code>!=</code>	<code>1 -ne 2</code>
-eq	<code>==</code>	<code>2 -eq (1+1)</code>
-le	<code><=</code>	<code>2 -le 2</code>
-lt	<code><</code>	<code>1 -lt 1</code>
-ge	<code>>=</code>	<code>Get-ChildItem Where-Object {\$_ .Length -ge 100}</code>
-gt	<code>></code>	<code>2 -gt 1</code>
-like		<code>Get-ChildItem Where-Object {\$_ .Name -like 'D*'}</code>
-notlike		<code>Get-ChildItem Where-Object {\$_ .Name -notlike 'D*'}</code>

Tabla 1. Comparadores lógicos. Fuente: elaboración propia.

Bucles

La expresión más habitual para recorrer un bucle es **ForEach**. Esta expresión ejecuta un bloque de código para todos los valores de una colección. Por ejemplo, el siguiente bloque imprime el nombre de todos los ficheros de una carpeta:

```
ForEach ($file in (Get-ChildItem -Path C:\Users\Administrator -File)) {
    Write-Host $file.Name
}
```

No es necesario inicializar la variable si **ForEach** recibe la colección a través de una tubería. En ese caso, la variable `$_.` contiene el valor de cada iteración (ya se vio en un ejemplo similar en la sección sobre `Get-Member`). Esta funcionalidad es especialmente útil en modo interactivo.

```
Get-ChildItem -Path C:\Users\Administrator -File | ForEach {
    Write-Host $_.Name
}
```

Las construcciones **While** y **For** funcionan como se esperaría:

```
$i = 1
While ($i -le 5) {
    Write-Host $i
    $i = $i + 1
}

For($j=1; $j -le 5; $j++)
{
    Write-Host $j
}
```

Tuberías

Como se ha mencionado anteriormente, las tuberías permiten **transferir objetos .NET entre comandos**. Dado que son objetos, se pueden manipular con algunos cmdlets específicos. Por ejemplo, Sort-Object facilita la ordenación de una colección a partir de atributos arbitrarios de los objetos. El siguiente ejemplo ordena los ficheros de un directorio por extensión y tamaño, de manera descendente.

```
PS C:\> Get-ChildItem -Path C:\Users\Administrator -File | Sort-Object -Property Extension,Length -Descending
```

Directory: C:\Users\Administrator

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
-a---	5/7/2020 2:53 AM	1148	hello-world.ps1
-a---	5/7/2020 2:54 AM	686	Write-Vars.ps1
-a---	5/6/2020 8:54 AM	80	datefolder.ps1
-a---	4/20/2020 12:37 AM	2693	security_key.ppk
-a---	4/19/2020 10:48 AM	397	putty.log

```
-a---- 5/6/2020 8:42 AM 13916 alias.csv
```

El comando **Where-Object**, que ya ha aparecido en algunos ejemplos, se usa para filtrar listas a partir de los atributos de los objetos.

```
PS C:\> Get-ChildItem -Path C:\Users\Administrator -File | Where-Object  
{$_Length -gt 100 -and $_.CreationTime -gt '5/1/2020'}
```

```
Directory: C:\Users\Administrator
```

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
-a---	5/6/2020 8:42 AM	13916	alias.csv
-a---	5/7/2020 2:53 AM	1148	hello-world.ps1
-a---	5/7/2020 2:54 AM	686	Write-Vars.ps1

El comando **Select-Object** se puede usar con dos objetivos: limitar el número de objetos de la colección y limitar las propiedades de estos objetos. La limitación en el número de objetos no es un filtrado como el de **Where-Object**, sino un truncado de la lista para obtener los primeros diez elementos, o los diez últimos, por ejemplo.

```
PS C:\> Get-ChildItem -Path C:\Users\Administrator -File | Sort-Object -  
Property Length -Descending | Select-Object -First 2 -Skip 1
```

```
Directory: C:\Users\Administrator
```

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
-a---	4/20/2020 12:37 AM	2693	security_key.ppk
-a---	5/7/2020 2:53 AM	1148	hello-world.ps1

Las propiedades se pueden seleccionar para, por ejemplo, guardar en un archivo CSV solo ciertas columnas.

```

PS C:\> Get-ChildItem -Path C:\Users\Administrator -File | Sort-Object -
Property Length -Descending | Select-Object -First 2 -Skip 1 -Property
Length,Name | Export-Csv output.csv
PS C:\Users\Administrator> Get-Content .\output.csv
#TYPE Selected.System.IO.FileInfo
"Length","Name"
"2693","security_key.ppk"
"1148","hello-world.ps1"

```

Y, por supuesto, estos comandos se pueden enlazar en una misma tubería arbitrariamente larga, como en la Figura 18.

```

PS C:\Users\Administrator> Get-ChildItem -Path C:\Users\Administrator -File |
>> Where-Object {$__.Length -gt 100 -and $__.CreationTime -gt '5/1/2020'} |
>> Sort-Object -Property Length -Descending |
>> Select-Object -First 2 -Skip 1 -Property Length,Name,CreationTime
Length Name CreationTime
-----
1148 hello-world.ps1 5/6/2020 11:40:28 PM
686 Write-Vars.ps1 5/7/2020 2:54:09 AM

```

Figura 18. Tubería con manipulación de objetos de una colección. Fuente: elaboración propia.

7.6. Referencias bibliográficas

Hill, K. (2009, marzo 8). *Effective Windows PowerShell: The Free eBook*. Keith Hill's Blog. <https://rkeithhill.wordpress.com/2009/03/08/effective-windows-powershell-the-free-ebook/>

Microsoft. (2021, marzo 22). *What is PowerShell?* <https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7>

Microsoft. (2021, junio 21). *About_Comparison_Operators*.
https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_comparison_operators?view=powershell-7

Shepard, M. (2015). *Getting Started with PowerShell*. Packt Publishing.

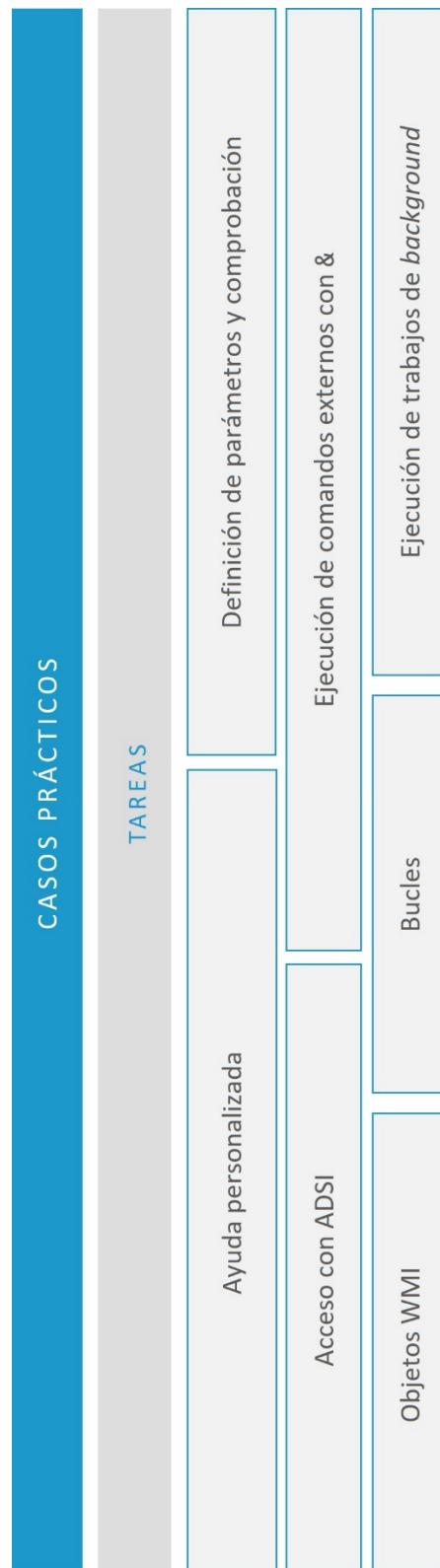
Administración de Sistemas en la Cloud

Automatización de instalación y configuración en Windows

Índice

Esquema	3
Ideas clave	4
8.1. Introducción y objetivos	4
8.2. Instalación de Directorio Activo	5
8.3. Creación de usuarios y grupos	17
8.4. Referencias bibliográficas	23

Esquema



8.1. Introducción y objetivos

Los comandos y sintaxis de PowerShell tiene poco valor, hasta que se ponen en un contexto más realista. Es necesario familiarizarse con ellos, pero no se aprecia su potencia hasta que no se integran en una tarea compleja.

En este tema, se presentan *scripts* de instalación de Directorio Activo. Uno de ellos configura el sistema, instala los roles y herramientas y crea el bosque y el primer dominio. El segundo creará unos pocos usuarios y grupos. Los *scripts* se analizarán paso a paso.

Los **objetivos** que se pretenden conseguir son:

- ▶ Tomar contacto con *scripts* complejos de PowerShell.
- ▶ Aprender a integrar *cmdlets* de PowerShell con otras herramientas de línea de comandos para construir *scripts* avanzados.

A continuación, en el vídeo *Paso a paso de instalación de Directorio Activo*, se demostrará, paso a paso, la ejecución de los *scripts* de este tema.



Accede al vídeo

8.2. Instalación de Directorio Activo

Active Directory se presentó como un servicio de directorio y de administración centralizada para equipos Windows. Una de las secciones era una guía de instalación en la que se explicaban algunos de los conceptos. El objetivo del primer *script* de este tema es automatizar esa instalación.

Se podría pensar que la creación de un bosque o de un dominio no debería automatizarse, porque es un momento muy crítico, o que no merece la pena, porque no ocurre de manera habitual. Sin embargo, también se puede argumentar a favor de esta **automatización**:

- ▶ Aunque este paso no ocurra a menudo en un entorno de producción, un *script* de este tipo permite acelerar la creación de entornos de prueba o de desarrollo, facilitando la tarea de desarrolladores y administradores.
- ▶ La automatización reduce el error humano, lo que la convierte en una herramienta ideal para un paso crítico como la creación del dominio.
- ▶ Un *script* puede incorporar controles y verificaciones que la instalación gráfica de Windows no tiene. Por ejemplo, una organización puede requerir que el nombre del dominio siga un formato concreto. También se podría integrar el *script* con una herramienta de gestión de contraseñas para almacenar la contraseña de recuperación de AD, evitando así que un administrador se olvidara o introdujera una errata al escribirla mal.

Script

A continuación, se muestra el *script* completo, listo para poder copiarlo a un fichero ps1. Los apartados siguientes comentarán cada una de las secciones.

```

<#
.SYNOPSIS
    Script de creacion de dominio
.DESCRIPTION
    Este script instalara los roles y caracteristicas de Windows
    necesarias para que el servidor funcione como un controlador de
    dominio. A continuacion creara un bosque con un dominio y reiniciara
    el equipo. Tambien configuara el firewall para permitir el
    trafico relacionado con AD.
.PARAMETER Dominio
    El nombre del nuevo dominio
.PARAMETER Dns1
    La IP del servidor DNS primario
.PARAMETER Dns2
    La IP del servidor DNS secundario
.PARAMETER Password
    La contraseña que se establecera para el usuario Administrator en el
    directorio activo (AD).
.INPUTS
    No espera datos por la entrada estandar.
.OUTPUTS
    No devuelve datos por la salida estandar.
.EXAMPLE
    PS C:/>instalar-ad.ps1 -dominio demo.loc -dns1 8.8.8.8 -dns2 8.8.4.4 -
password secreto
.NOTES

```

#>

```

param (
    [string]$Dominio = "demo.loc",
    [string]$Dns1 = "8.8.8.8",
    [string]$Dns2 = "8.8.4.4",
    [Parameter(Mandatory=$true)][string]$Password
)

# inicializar y verificar los componentes del nombre de dominio
$componentes = $Dominio.Split(".")
If ($componentes.Length -ne 2) {
    Write-Host "El nombre de dominio debe tener el formato NOMBRE.SUFIJO"
}

```

```

        Exit 1
    }

$netbios, $suffix = $componentes[0], $componentes[1]

# guardar la contraseña como un objeto seguro
$PassSegura = ConvertTo-SecureString -String "$Password" `

    -AsPlainText -Force

# cambiar la contraseña del administrador
$admin = [adsi]('WinNT://./administrator, user')
$admin.psbase.invoke('SetPassword', $Password)

# configurar reglas de firewall necesarias
$reglas = ("Remote Administration",
            "File and Printer Sharing",
            "Remote Service Management",
            "Performance Logs and Alerts",
            "Remote Event Log Management",
            "Remote Volume Management",
            "Remote Scheduled Tasks Management",
            "Remote Desktop",
            "Windows Firewall Remote Management")

ForEach ($regla in $reglas) {
    & netsh advfirewall firewall set rule group="$regla" new enable =yes
}

# configurar los servidores dns en la NIC
$nicConfig = Get-WmiObject Win32_NetworkAdapterConfiguration `

    -Filter "ipenabled = 'true'" | Select-Object -First 1
$nic = Get-WmiObject Win32_NetworkAdapter `

    -Filter "InterfaceIndex = $($nicConfig.InterfaceIndex)"
& netsh interface ip set dnsservers name="$(($nic.NetConnectionID))" `

    source=static address=$Dns1
& netsh interface ip add dnsservers name="$(($nic.NetConnectionID))" `

    address=$Dns2 index=2
# instalar los servicios y herramientas del directorio activo
Start-Job -Name addFeature -ScriptBlock {

    Add-WindowsFeature "RSAT-AD-Tools"
}

```

```

$features = ("AD-Domain-Services", "DNS", "GPMC")
ForEach ($feature in $features) {
    Add-WindowsFeature -Name $feature -IncludeAllSubFeature ` 
        -IncludeManagementTools
}
}

Wait-Job -Name addFeature
Receive-Job -Name addFeature
if (! $?) {
    Write-Host "Ha ocurrido un error durante la instalacion de los roles"
    Exit 2
}

# crear el bosque y el dominio
Start-Job -Name addForest -ArgumentList $Dominio, $PassSegura, $netbios ` 
-ScriptBlock {
    Import-Module ADDSDeployment
    Install-ADDSForest -DomainName $args[0] ` 
        -SafeModeAdministratorPassword $args[1] ` 
        -DomainNetbiosName $args[2] ` 
        -DomainMode Default -ForestMode Default ` 
        -DatabasePath "%SYSTEMROOT%\NTDS" -LogPath "%SYSTEMROOT%\NTDS" ` 
        -SysvolPath "%SYSTEMROOT%\SYSVOL" ` 
        -InstallDns -Force
}
Wait-Job -Name addForest
Receive-Job -Name addForest
if (! $?) {
    Write-Host "Ha ocurrido un error durante la creacion del bosque"
    Exit 3
}

```

Ayuda interactiva

El *script* arranca con el comentario con formato especial, que PowerShell interpretará como la ayuda del *script*.

```
<#
.SYNOPSIS
    Script de creacion de dominio
.DESCRIPTION
    Este script instalara los roles y caracteristicas de Windows
    necesarias para que el servidor funcione como un controlador de
    dominio. A continuacion creara un bosque con un dominio y reiniciara
    el equipo. Tambien configuara el firewall para permitir el
    trafico relacionado con AD.
.PARAMETER Dominio
    El nombre del nuevo dominio
.PARAMETER Dns1
    La IP del servidor DNS primario
.PARAMETER Dns2
    La IP del servidor DNS secundario
.PARAMETER Password
    La contraseña que se establecerá para el usuario Administrator en el
    directorio activo (AD).
.INPUTS
    No espera datos por la entrada estandar.
.OUTPUTS
    No devuelve datos por la salida estandar.
.EXAMPLE
    PS C:/>instalar-ad.ps1 -dominio demo.loc -dns1 8.8.8.8 -dns2 8.8.4.4 -
    password secreto
.NOTES
#>
```

Al igual que con un *cmdlets*, Get-Help mostrará la ayuda al pasar el nombre del *script* como parámetro (ver Figura 1).

```

PS C:\temp> Get-Help .\install.ps1 -Detailed

NAME
    C:\temp\install.ps1

SYNOPSIS
    Script de creacion de dominio

SYNTAX
    C:\temp\install.ps1 [[-Dominio] <String>] [[-Dns1] <String>] [[-Dns2] <String>] [-Password] <String>
    [<CommonParameters>]

DESCRIPTION
    Este script instalara los roles y caracteristicas de Windows
    necesarias para que el servidor funcione como un controlador de
    dominio. A continuacion creara un bosque con un dominio y reiniciara
    el equipo. Tambien configura el firewall para permitir el
    trafico relacionado con AD.

PARAMETERS
    -Dominio <String>
        El nombre del nuevo dominio

    -Dns1 <String>
        La IP del servidor DNS primario

    -Dns2 <String>
        La IP del servidor DNS secundario

    -Password <String>
        La contraseña que se establecera para el usuario Administrator en el directorio activo (AD).

    <CommonParameters>
        This cmdlet supports the common parameters: Verbose, Debug,
        ErrorAction, ErrorVariable, WarningAction, WarningVariable,
        OutBuffer, PipelineVariable, and OutVariable. For more information, see
        about_CommonParameters (https://go.microsoft.com/fwlink/?LinkID=113216).

----- EXAMPLE 1 -----
PS C:/>instalar-ad.ps1 -dominio demo.loc -dns1 8.8.8.8 -dns2 8.8.4.4 -password secreto

```

Figura 1. Ayuda integrada del *script*. Fuente: elaboración propia.

Parámetros

PowerShell ofrece una funcionalidad integrada para el «parseo» de parámetros. Este *script* solo necesita parámetros de tipo cadena, pero PowerShell se encargaría de verificar el tipo si alguno de ellos fuera de tipo entero.

```

param (
    [string]$Dominio = "demo.loc",
    [string]$Dns1 = "8.8.8.8",
    [string]$Dns2 = "8.8.4.4",
    [Parameter(Mandatory=$true)][string]$Password
)

```

La verificación de tipo no tiene por qué validar los valores completamente. Por ejemplo, los nombres de dominio deben seguir el esquema de nombres DNS. Se podría obviar la comprobación de este formato y dejar que la creación del dominio

falle más adelante. Sin embargo, se puede mejorar la experiencia del usuario si el *script* detecta el problema al principio.

En este caso concreto, el formato se puede verificar dividiendo el parámetro \$dominio entre los puntos de la cadena de texto, de manera que, si el \$dominio es "demo.loc", \$componentes es una lista con dos cadenas, "demo" y "loc". El nombre del dominio no tiene por qué estar limitado a dos componentes, pero una organización podría establecer este requisito como una política interna.

```
# inicializar y verificar los componentes del nombre de dominio
$componentes = $Dominio.Split(".")
If ($componentes.Length -ne 2) {
    Write-Host "El nombre de dominio debe tener el formato NOMBRE.SUFIJO"
    exit 3
}
```

Las siguientes líneas manipulan los parámetros para construir variables necesarias más adelante. El nombre del dominio completo es necesario tal cual, pero, en algunos casos, es necesario usar los componentes por separado. Además, las contraseñas deben almacenarse como un objeto SecureString.

```
$netbios, $suffix = $componentes[0], $componentes[1]

# guardar la contraseña como un objeto seguro
$PassSegura = ConvertTo-SecureString -String "$Password" `

-AsPlainText -Force
```

Este *script* intenta ser 100 % automatizable, pero en una ejecución interactiva se podría haber optado por leer la contraseña por la línea de comandos (ver Figura 2). Este paso requiere una acción del usuario, por lo que no es ideal para un *script* de automatización.

```
PS C:\temp> $SecureInput = Read-Host -AsSecureString  
*****  
PS C:\temp> $SecureInput  
System.Security.SecureString
```

Figura 2. Contraseña como cadena segura. Fuente: elaboración propia.

Cambio de la contraseña local

Este paso modifica la contraseña del administrador al valor introducido como parámetro. La cuenta de administrador local se convertirá en administrador del dominio, por lo que este paso es parte de la configuración de AD. No es estrictamente necesario, pero se incluye como ejemplo de una las integraciones de PowerShell.

PowerShell no solo permite el uso de *cmdlets* o de binarios. Este ejemplo hace uso de una integración con un objeto [COM](#), una interfaz de intercambio de objetos entre procesos propia de Microsoft. Aunque es muy antigua, sigue estando disponible y, aunque suele haber comandos de PowerShell equivalentes, el uso de esta interfaz permite que las organizaciones reutilicen *scripts* antiguos sin necesidad de reescribirlos como comandos nativos de PowerShell.

La interfaz COM del ejemplo es [ADSI](#). Aunque está pensada para acceder a objetos de AD, en este caso se conecta al equipo local (en este punto del *script* aún no se ha creado el dominio). La cadena "WinNT://./administrator" indica una conexión al objeto administrator del equipo local, identificado por un «.».

```
# cambiar la contraseña del administrador  
$admin = [adsi]('WinNT://./administrator, user')  
$admin.psbase.invoke('SetPassword', $Password)
```

Configuración del *firewall*

Al igual que la contraseña del administrador, la configuración del *firewall* de Windows no es estrictamente una parte necesaria de la creación del dominio. Sirve, no obstante, para mostrar dos funcionalidades como ejemplo.

Por un lado, el *script* hace uso de un bucle *foreach* para ejecutar el mismo comando tantas veces como reglas hay que configurar. De esa manera, si hay que cambiar el comando, solo es necesario editar una línea.

La segunda funcionalidad es la ejecución de comandos arbitrarios desde PowerShell. El *script* usa [netsh](#), una utilidad de configuración de interfaces de red para Windows. PowerShell dispone de varias opciones para arrancar programas externos. Una de ellas es Start-Process. La que usa el *script* es el símbolo «&», que ejecuta el resto de la línea en una consola Windows, o cmd.exe, no en una consola de PowerShell.

PowerShell dispone de *cmdlets* para configurar el *firewall*, tales como Set-NetFirewallRule y New-NetFirewallRule, que podrían usarse nativamente en el *script*. Este bloque debe considerarse únicamente como un ejemplo.

```
# configurar reglas de firewall necesarias
$reglas = ("Remote Administration",
            "File and Printer Sharing",
            "Remote Service Management",
            "Performance Logs and Alerts",
            "Remote Event Log Management",
            "Remote Volume Management",
            "Remote Scheduled Tasks Management",
            "Remote Desktop",
            "Windows Firewall Remote Management")

ForEach ($regla in $reglas) {
    & netsh advfirewall firewall set rule group="$regla" new enable =yes
}
```

Servidores DNS

El siguiente bloque configura los servidores DNS como valores estáticos, la configuración recomendada para AD. El *script* vuelve a hacer uso de netsh para la configuración, pero usa la interfaz [WMI](#) para recuperar el ID de la tarjeta de red. WMI es una interfaz para la administración de múltiples objetos del sistema operativo. No es una interfaz exclusiva de PowerShell; de hecho, era un método muy común en VBScript, antes de que se extendiera el uso de PowerShell.

```
# configurar los servidores dns en la NIC
$nicConfig = Get-WmiObject Win32_NetworkAdapterConfiguration ` 
    -Filter "ipenabled = 'true'" | Select-Object -First 1
$nic = Get-WmiObject Win32_NetworkAdapter ` 
    -Filter "InterfaceIndex = $($nicConfig.InterfaceIndex)"
& netsh interface ip set dnsservers name="$(($nic.NetConnectionID))" ` 
    source=static address=$Dns1
& netsh interface ip add dnsservers name="$(($nic.NetConnectionID))" ` 
    address=$Dns2 index=2
```

Instalación de roles y características

En este punto del *script*, la configuración preliminar ya está lista. El siguiente paso consiste en la instalación de los servicios y herramientas de administración necesarios. Las llamadas al cmdlet Add-WindowsFeature se podrían usar directamente, ya que son síncronas (es decir, bloquean la consola sin poder progresar hasta que terminan), pero el *script* hace uso de Start-Job para demostrar el uso de trabajos **de fondo** (o de *background*).

El cmdlet Start-Job arranca un proceso de PowerShell adicional, independiente del proceso que ejecuta el *script*. Ese segundo proceso ejecuta el código del bloque indicado por el parámetro ScriptBlock. No tiene acceso al ámbito del proceso principal, por lo que no puede usar las variables definidas en el *script* principal. A todos los efectos, es igual que arrancar un programa y dejar que se ejecute.

Una vez arrancado, los *cmdlets* Wait-Job y Receive-Job se encargan de esperar a que el trabajo termine y de recibir la salida del trabajo, si la hubiera, respectivamente. Si el trabajo ha terminado satisfactoriamente, el *script* principal continuará su ejecución. Por el contrario, si el trabajo terminó con error, la variable automática «\$?» contendrá False y el *script* terminará con un mensaje al usuario y un código de error específico, Exit 2. La variable «\$?» es la misma que en Bash contiene el código de salida, pero, mientras que en Bash es un entero, en PowerShell es un booleano.

```
# instalar los servicios y herramientas del directorio activo
Start-Job -Name addFeature -ScriptBlock {
    Add-WindowsFeature "RSAT-AD-Tools"

    $features = ("AD-Domain-Services", "DNS", "GPMC")
    ForEach ($feature in $features) {
        Add-WindowsFeature -Name $feature -IncludeAllSubFeature `

            -IncludeManagementTools
    }
}
Wait-Job -Name addFeature
Receive-Job -Name addFeature
if (! $?) {
    Write-Host "Ha ocurrido un error durante la instalacion de los roles"
    Exit 2
}
```

Creación del bosque y del primer dominio

El último bloque sigue el mismo esquema que el anterior al crear un trabajo para ejecutar un bloque de código. Sin embargo, en este caso, el bloque de código necesita acceder a las variables del *script* principal. Para ello, el *cmdlet* Start-Job acepta una lista de parámetros que el bloque recibirá en la variable \$args. Estos argumentos ya no tienen nombre, por lo que hay que acceder a ellos con el orden en el que se pasan a ArgumentList.

Este paso, además, muestra el uso de Import-Module. PowerShell es extensible mediante módulos, pero estos módulos no tienen por qué estar cargados por defecto en cualquier consola. La Figura 3 muestra los módulos importados por defecto en una sesión (esto puede variar de un equipo a otro, en función de los roles instalados). Una vez importado un módulo, los cmdlets definidos por este se pueden usar en el resto de la sesión.

PS C:\temp> Get-Module			
ModuleType	Version	Name	ExportedCommands
Binary	1.0.0.0	CimCmdlets	{Export-BinaryMiLog, ...}
Manifest	3.1.0.0	Microsoft.PowerShell.Management	{Add-Computer, Add-C...}
Manifest	3.0.0.0	Microsoft.PowerShell.Security	{ConvertFrom-SecureS...}
Manifest	3.1.0.0	Microsoft.PowerShell.Utility	{Add-Member, Add-Typ...}
Manifest	3.0.0.0	Microsoft.WSMAN.Management	{Connect-WSMan, Disa...}
Manifest	2.0.0.0	NetSecurity	{Get-DAPolicyChange, ...}
Script	2.0.0	PSReadline	{Get-PSReadLineKeyHa...}
Binary	1.1.0.0	PSScheduledJob	{Add-JobTrigger, Dis...
Manifest	2.0.0.0	PSWorkflow	{New-PSWorkflowExecu...}
Script	2.0.0.0	ServerManager	{Get-WindowsFeature, ...}

Figura 3. Módulos importados por defecto. Fuente: elaboración propia.

El cmdlet Install-ADDSForest es similar y acepta los mismos parámetros.

```
# crear el bosque y el dominio
Start-Job -Name addForest -ArgumentList $Dominio, $PassSegura, $netbios ` 
-ScriptBlock {
    Import-Module ADDSDeployment
    Install-ADDSForest -DomainName $args[0] ` 
        -SafeModeAdministratorPassword $args[1] ` 
        -DomainNetbiosName $args[2] ` 
        -DomainMode Default -ForestMode Default ` 
        -DatabasePath "%SYSTEMROOT%\NTDS" -LogPath "%SYSTEMROOT%\NTDS" ` 
        -SysvolPath "%SYSTEMROOT%\SYSVOL" ` 
        -InstallDns -Force
}
Wait-Job -Name addForest
Receive-Job -Name addForest
if (! $?) {
    Write-Host "Ha ocurrido un error durante la creacion del bosque"
    Exit 3
}
```

Una vez completada la instalación, el equipo se reiniciará y ya será necesario iniciar sesión con la cuenta de administrador de dominio.

8.3. Creación de usuarios y grupos

El siguiente *script* sirve de ejemplo de creación de recursos de Directorio Activo. Se ha separado del *script* anterior, ya que solo tiene sentido crear recursos una vez que existe el dominio.

```
<#
.SYNOPSIS
    Script de creacion de usuarios y grupos de AD
.DESCRIPTION
    Este script instalara inicializa un dominio de AD con varios usuarios
    de prueba, una OU nueva para grupos, y varios grupos en esa OU a
    los que agregara los usuarios como miembros. El script espera dos
    contraseñas: la del administrador de dominio para crear la OU
    y la contraseña por defecto de los usuarios.
.PARAMETER Dominio
    El nombre del dominio
.PARAMETER PasswordAdmin
    La contraseña del Administrador del dominio.
.PARAMETER PasswordUsuarios
    La contraseña por defecto de los nuevos usuarios.
.EXAMPLE
    PS C:/>crear-grupos.ps1 -Dominio demo.loc -PasswordAdmin secreto1 -
    PasswordUsuarios secreto2
.NOTES
#>

param (
    [string]$Dominio = "demo.loc",
    [Parameter(Mandatory=$true)][string]$PasswordAdmin,
    [Parameter(Mandatory=$true)][string]$PasswordUsuarios
```

```

)

# inicializar y verificar los componentes del nombre de dominio
$componentes = $Dominio.Split(".")
If ($componentes.Length -gt 2) {
    Write-Host "El nombre de dominio debe tener el formato NOMBRE.SUFIJO"
    exit 3
}
$netbios, $suffix = $componentes[0], $componentes[1]

# guardar la contraseña como un objeto seguro
$PassSeguraAdmin = ConvertTo-SecureString -String "$PasswordAdmin" `

-AsPlainText -Force

$PassSeguraUser = ConvertTo-SecureString -String "$PasswordUsuarios" `

-AsPlainText -Force

# crear la unidad organizativa
$credencial = New-Object
System.Management.Automation.PSCredential("$netbios\Administrator",
$PassSeguraAdmin)
New-ADOrganizationalUnit -Credential $credencial -Name Groups `

-Path "DC=$netbios,DC=$suffix"

# crear 5 usuarios de prueba
$a = 1
Do
{
    "Creando el usuario de pruebas: Tester_$a"
    New-ADUser -Credential $credencial -Name Tester-$a `

    -Path "CN=Users,DC=$netbios,DC=$suffix" `

    -SamAccountName admin-tester-$a `

    -GivenName Tester-$a -SurName Test-Tested-$a `

    -AccountPassword $PassSeguraUser -Description Testing-User-$a `

    -UserPrincipalName admin-tester-$a@$Dominio `

    -DisplayName Tester-Test-Tested-$a `

    -PasswordNeverExpires $false -Enabled $true
    $a++
} While ($a -le 5)

```

```

# crear 3 grupos y añadir todos los usuarios a todos los grupos
$g = 1
Do
{
    New-ADGroup -Credential $credencial -Name "Administrators_$g" ` 
        -GroupScope Global ` 
        -Path "OU=Groups,DC=$netbios,DC=$suffix" 2>&1 | out-null
    $a = 1
    Do
    {
        "Agregando el usuario Tester_$a al grupo Administrators_$g"
        Add-ADGroupMember -Credential $credencial ` 
            -identity "CN=Administrators_$g,OU=Groups,DC=$netbios,DC=$suffix" ` 
            -Members "CN=Tester-$a,CN=Users,DC=$netbios,DC=$suffix" 2>&1 | out-
null
        $a++
    } While ($a -le 5)
    $g++
} While ($g -le 3)

```

Inicialización y parámetros

Estos bloques son muy parecidos a los del apartado anterior, por lo que no se entrará en detalle. El único apartado relevante es que el *script acepta dos contraseñas*: la contraseña del administrador que ejecuta el *script* y la contraseña por defecto que tendrán los usuarios.

En un caso real, sería habitual que el *script* de creación de cuentas generara una contraseña temporal aleatoria que se hiciera llegar al usuario por algún mecanismo (un *email*, si el usuario dispone de buzón, o un *email* a su responsable, por ejemplo). En ese caso, el administrador no llegaría nunca a conocer la primera contraseña del usuario.

Creación de OU

El primer paso consiste en crear la unidad organizativa que contendrá las cuentas del grupo. El cmdlet `New-ADOrganizationalUnit` requiere un tipo de objeto `PSCredential` para la contraseña, diferente a los del anterior *script*.

```
# crear la unidad organizativa
$credencial = New-Object
System.Management.Automation.PSCredential("$netbios\Administrator",
$PassSeguraAdmin)
New-ADOrganizationalUnit -Credential $credencial -Name Groups ` 
-Path "DC=$netbios,DC=$suffix"
```

Creación de usuarios

El ejemplo de creación de usuarios no tiene por qué ser el más útil en un entorno real, pero puede ser habitual en entornos de prueba, donde hace falta generar varias cuentas de usuario de manera rápida. El bloque del bucle, un *do-while*, inicializa muchos parámetros de la cuenta del usuario a partir de una variable índice.

```
# crear 5 usuarios de prueba
$a = 1
Do
{
    "Creando el usuario de pruebas: Tester_$a"
    New-ADUser -Credential $credencial -Name Tester-$a ` 
-Path "CN=Users,DC=$netbios,DC=$suffix" ` 
-SamAccountName admin-tester-$a ` 
-GivenName Tester-$a -SurName Test-Tested-$a ` 
-AccountPassword $PassSeguraUser -Description Testing-User-$a ` 
-UserPrincipalName admin-tester-$a@$Dominio ` 
-DisplayName Tester-Test-Tested-$a ` 
-PasswordNeverExpires $false -Enabled $true
    $a++
} While ($a -le 5)
```

Creación de grupos

El último bloque sigue un esquema similar al anterior, pero con dos bucles anidados: el externo crea los grupos y el interno añade todos los usuarios como miembros del grupo.

La salida de los comandos `New-ADGroup` y `Add-ADGroupMember` envía tanto la salida estándar como la salida de errores a `Out-Null`. Este *cmdlet* es similar al dispositivo `/dev/null` de Linux: borra los datos, en vez de redirigirlos por la tubería. El objetivo es evitar que aparezca un error si ya existe un grupo o si un miembro ya ha sido añadido.

```
# crear 3 grupos y añadir todos los usuarios a todos los grupos
$g = 1
Do
{
    New-ADGroup -Credential $credencial -Name "Administrators_$g" `

        -GroupScope Global `

        -Path "OU=Groups,DC=$netbios,DC=$suffix" 2>&1 | out-null
    $a = 1
    Do
    {
        "Agregando el usuario Tester_$a al grupo Administrators_$g"
        Add-ADGroupMember -Credential $credencial `

            -identity "CN=Administrators_$g,OU=Groups,DC=$netbios,DC=$suffix" `

            -Members "CN=Tester-$a,CN=Users,DC=$netbios,DC=$suffix" 2>&1 | out-
null
        $a++
    } While ($a -le 5)
    $g++
} While ($g -le 3)
```

Además, tanto este bloque como el anterior incluyen cadenas de texto sin un *cmdlet* a la vista. El comportamiento de PowerShell es escribir estos datos por la salida estándar, de manera similar a volcarlos con `Write-Host`. El resultado es igual a una línea de *log* (ver Figura 4).

```

PS C:\temp> .\create.ps1 -Dominio demo.loc -PasswordAdmin Password1! -PasswordUsuarios Prueba1234!
Creando el usuario de pruebas: Tester_1
Creando el usuario de pruebas: Tester_2
Creando el usuario de pruebas: Tester_3
Creando el usuario de pruebas: Tester_4
Creando el usuario de pruebas: Tester_5
Agregando el usuario Tester_1 al grupo Administrators_1
Agregando el usuario Tester_2 al grupo Administrators_1
Agregando el usuario Tester_3 al grupo Administrators_1
Agregando el usuario Tester_4 al grupo Administrators_1
Agregando el usuario Tester_5 al grupo Administrators_1
Agregando el usuario Tester_1 al grupo Administrators_2
Agregando el usuario Tester_2 al grupo Administrators_2
Agregando el usuario Tester_3 al grupo Administrators_2
Agregando el usuario Tester_4 al grupo Administrators_2
Agregando el usuario Tester_5 al grupo Administrators_2
Agregando el usuario Tester_1 al grupo Administrators_3
Agregando el usuario Tester_2 al grupo Administrators_3
Agregando el usuario Tester_3 al grupo Administrators_3
Agregando el usuario Tester_4 al grupo Administrators_3
Agregando el usuario Tester_5 al grupo Administrators_3
PS C:\temp>

```

Figura 4. Ejecución del *script* de creación de usuarios y grupos. Fuente: elaboración propia.

Comprobación

El hecho de poder iniciar sesión en el equipo tras el reinicio es prueba suficiente de que el primer *script* funcionó como se esperaba y, si el segundo *script* termina sin errores, es muy probable que las cuentas de usuario ya estén disponibles. Se podría comprobar en la consola gráfica, abriendo el panel de Active Directory Users and Computers desde el Server Manager, tal como muestra la Figura 5.

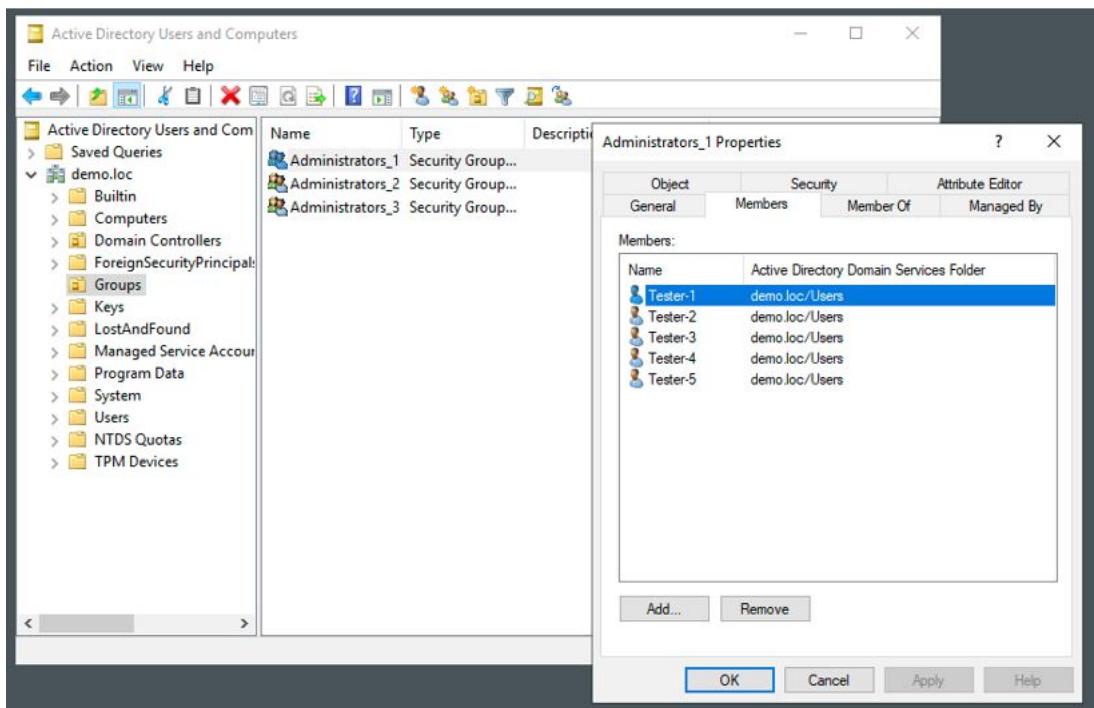


Figura 5. Consola de AD con los usuarios recién creados. Fuente: elaboración propia.

También es posible, dado que el tema trata sobre PowerShell, consultar el Directorio Activo directamente en la línea de comandos:

```
PS C:\> $ou = (Get-ADOrganizationalUnit -Filter 'Name -Like "Groups"')  
PS C:\> $ou.Name  
Groups  
  
PS C:\> Get-ADGroup -Filter * -SearchBase $ou.DistinguishedName | Select-Object -Property SamAccountName, SID  
  
SamAccountName      SID  
-----  
Administrators_1    S-1-5-21-3436996036-797900140-2994447757-1126  
Administrators_2    S-1-5-21-3436996036-797900140-2994447757-1127  
Administrators_3    S-1-5-21-3436996036-797900140-2994447757-1128
```

8.4. Referencias bibliográficas

Microsoft. (2018, mayo 31). *ADSI Interfaces*. <https://docs.microsoft.com/en-us/windows/win32/adsi/adsi-interfaces>

Microsoft. (2018, mayo 31). *Component Object Model (COM)*.
<https://docs.microsoft.com/en-gb/windows/win32/com/component-object-model-com--portal?redirectedfrom=MSDN>

Microsoft. (2018, mayo 31). *Windows Management Instrumentation*.
<https://docs.microsoft.com/en-us/windows/win32/wmisdk/wmi-start-page>

Microsoft. (2020, julio 8). *Netsh Command Syntax, Contexts, and Formatting*.
<https://docs.microsoft.com/en-us/windows-server/networking/technologies/netsh/netsh-contexts>

Shepard, M. (2015). *Getting Started with PowerShell*. Packt Publishing.

Administración de Sistemas en la Cloud

Administración básica de sistema operativo en Windows

Índice

Esquema	3
Ideas clave	4
9.1. Introducción y objetivos	4
9.2. Directivas de grupos	5
9.3. PowerShell remota	11
9.4. Herramientas de sistema	14
9.5. Referencias bibliográficas	20

Esquema

ADMINISTRACIÓN BÁSICA DE SISTEMA OPERATIVO EN WINDOWS

Directivas de grupo

- ▶ Personalizan configuraciones de Windows
- ▶ Se agrupan en GPO
- ▶ Se aplican a nivel local, de sitio, de dominio y de OU

PowerShell remota

- ▶ Get-Service schedule -ComputerName server1
- ▶ Enter-PSSession -ComputerName server1

Herramientas del sistema

- ▶ Visor de eventos
- ▶ Administrador de tareas
- ▶ Programador de tareas
- ▶ Suite de herramientas Sysinternals

9.1. Introducción y objetivos

Este tema profundiza en algunas de las herramientas necesarias para la administración de equipos Windows. Las directivas de grupo son la opción más extendida para aplicar configuraciones en grandes flotas de servidores y equipos de escritorio. En este se muestra en detalle cómo usar PowerShell remotamente. Para finalizar, se citan unas cuantas herramientas, que son la navaja suiza de cualquier administrador de equipos Windows.

Los comandos y sintaxis de PowerShell tienen poco valor, hasta que se ponen en un contexto más realista. Es necesario familiarizarse con ellos, pero no se aprecia su potencia hasta que no se integran en una tarea compleja.

Los **objetivos** que se pretenden conseguir son:

- ▶ Entender el concepto de directiva de grupo y su uso en Directorio Activo.
- ▶ Aprender el uso de PowerShell como herramienta de administración remota.
- ▶ Tomar un primer contacto con varias de las herramientas habituales de administración en Windows.

A continuación, en el vídeo *Administración de usuarios y procesos en Windows*, se explicará brevemente la consola de administración de directivas de grupo y algunas de las opciones disponibles para administrar servicios y procesos.



Accede al vídeo

9.2. Directivas de grupos

Una de las fortalezas de los sistemas operativos basados en Windows es su **flexibilidad**. Sin embargo, esta flexibilidad tiene un precio: en general, los usuarios sin privilegios administrativos de una red no deberían ser capaces de cambiar muchas de las configuraciones del sistema, como la configuración de TCP/IP y las políticas de seguridad de contraseñas.

Las **directivas de grupo**, o *group policies*, también denominadas GPO (*group policy object*), están diseñadas para proporcionar a los administradores del sistema la capacidad de personalizar la configuración del usuario final y establecer restricciones sobre los tipos de acciones que los usuarios pueden realizar. Los administradores pueden crear directivas de grupo y luego aplicarlas a uno o más usuarios, servidores o equipos de escritorio dentro del entorno. En general, estas configuraciones modifican opciones del registro de Windows, pero es más fácil configurar las opciones mediante directivas de grupo que realizar cambios en el registro manualmente.

Funcionamiento de las directivas de grupo

La configuración de las directivas se basa en las **plantillas administrativas de la directiva de grupo**. Estas plantillas proporcionan una lista de opciones de configuración. Por ejemplo, una opción para un usuario o equipo es Do not keep history of recently opened documents (no mantener un histórico de los documentos recientes, ver Figura 1). Cuando se establece la opción, se realiza el cambio apropiado en el registro de Windows de las sesiones de los usuarios y los equipos a los que se aplica la directiva.

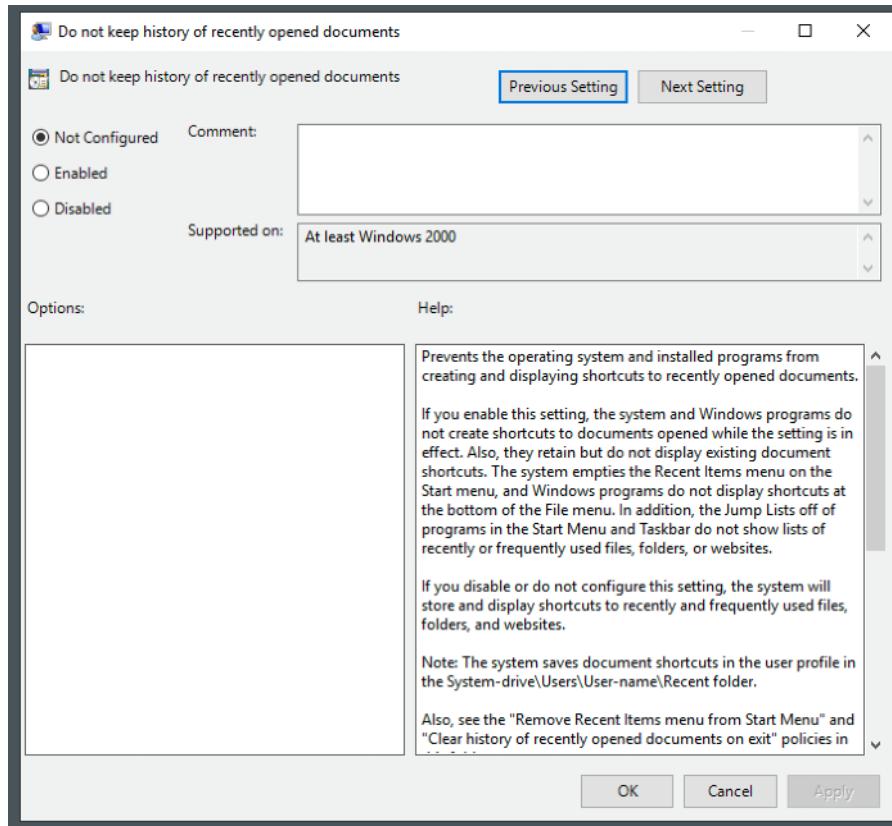


Figura 1. Directiva que modifica cómo se comporta el histórico de documentos recientes. Fuente: elaboración propia.

De manera predeterminada, Windows viene con varias plantillas administrativas. Además, los administradores y los desarrolladores de aplicaciones pueden crear sus propios archivos de plantilla administrativa para una funcionalidad específica.

La mayoría de los elementos de la directiva de grupo tienen tres opciones de configuración diferentes (ver Figura 1):

- ▶ Enabled: especifica que se ha establecido una configuración para esta GPO. Algunas GPO requieren que se establezcan valores u opciones concretos. Por ejemplo, la directiva de *account lockout* (bloqueo de cuenta) debe especificar cuántos intentos de inicio de sesión incorrectos pueden realizarse antes de que la cuenta se bloquee.
- ▶ Disabled. Especifica que esta opción está desactivada, es decir, que el administrador del sistema desea no permitir ciertas funciones.

- ▶ Not Configured. Especifica que esta configuración no se ha habilitado ni deshabilitado. Simplemente establece que esta directiva de grupo no establece esta configuración, aunque otras GPO pueden haberla configurado.

La configuración de la directiva de grupo puede aplicarse a dos tipos de objetos: usuarios y equipos, tanto locales como de Directorio Activo. Tanto los usuarios como los equipos pueden organizarse en unidades organizativas y las GPO pueden aplicarse precisamente a nivel de OU, por lo que este tipo de configuración simplifica la administración de un número arbitrario de objetos con una única GPO.

Las principales opciones que puede configurar dentro de las directivas de grupo son las siguientes:

- ▶ **Configuración de software** (*software settings*). Se aplican a aplicaciones y software específicos que pueden instalarse en los equipos. Los administradores pueden usar esta configuración para hacer que las nuevas aplicaciones estén disponibles para los usuarios finales y para controlar la configuración predeterminada de estas.
- ▶ **Configuración de Windows** (*Windows settings*). Permiten a los administradores personalizar el comportamiento del sistema operativo. Incluye opciones para configurar Internet Explorer (incluida la página de inicio predeterminada y otras configuraciones), seguridad (como la política de cuenta) o el registro de eventos.
- ▶ **Plantillas administrativas** (*administrative templates*). Se utilizan para configurar aún más las configuraciones de usuario y equipo. Además de las opciones predeterminadas disponibles, los administradores del sistema pueden crear sus propias plantillas administrativas con opciones personalizadas.

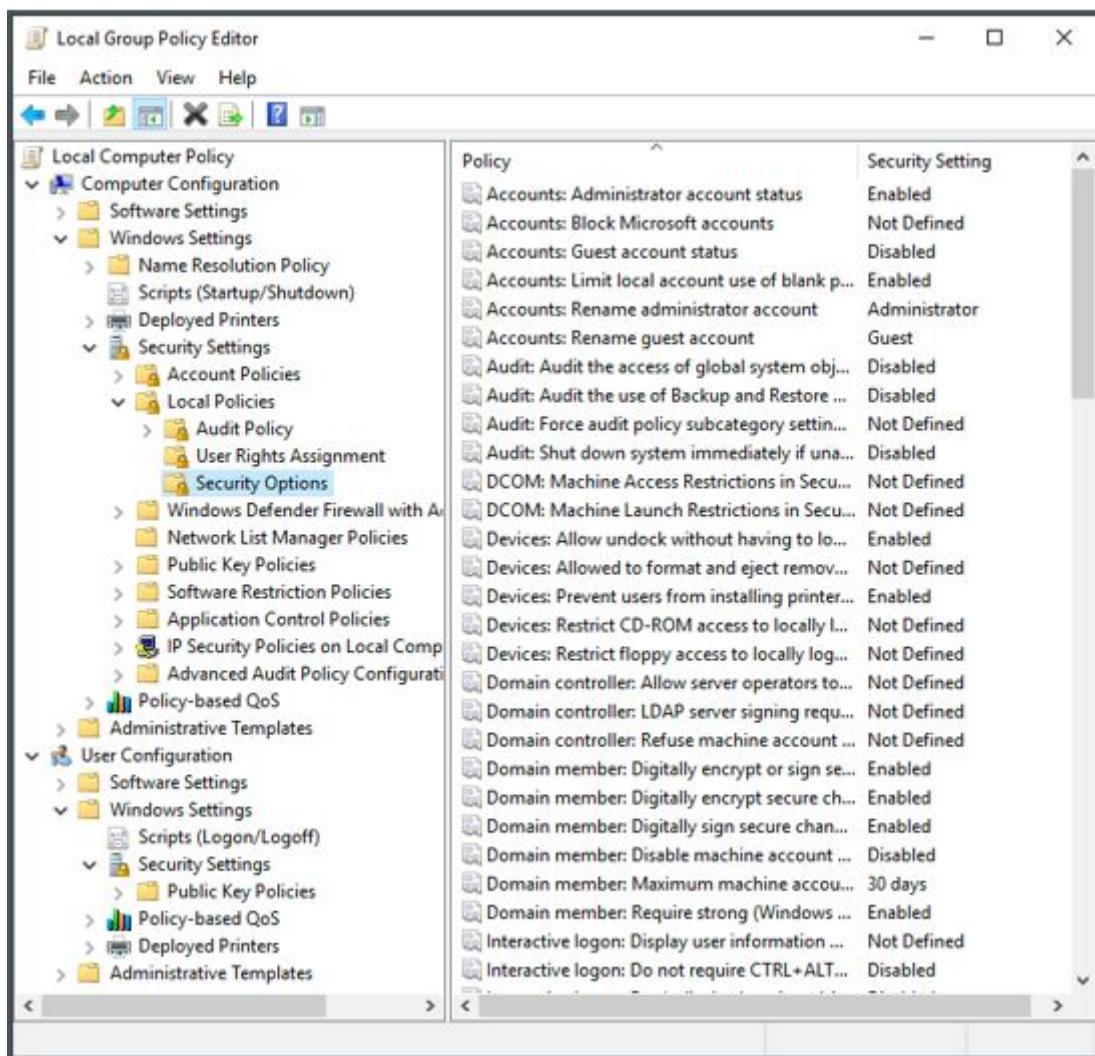


Figura 2. Opciones de seguridad en una directiva de equipo. Fuente: elaboración propia.

Configurar las directivas

Para que sean más fáciles de administrar, las directivas de grupo se agrupan en los ya citados objetos de directiva de grupo o GPO. Los GPO actúan como contenedores de directivas, lo que simplifica la administración de estas. Por ejemplo, un administrador puede tener diferentes políticas para usuarios y equipos en diferentes departamentos. Según estos requisitos, puede crear un GPO para los miembros del departamento de ventas y otro para los miembros del departamento de ingeniería. Luego, podría aplicar los GPO a la unidad organizativa para cada departamento.

Otro concepto importante es que la configuración de la directiva de grupo es jerárquica. Hay cuatro niveles que determinan la prioridad de procesamiento de GPO:

- ▶ Local. Cada equipo tiene un objeto de directiva de grupo que se almacena localmente.
- ▶ Sitios. La configuración de GPO de un sitio se aplica a todos los dominios y servidores que forman parte de este.
- ▶ Dominios. Los dominios son el tercer nivel al que los administradores pueden asignar GPO. La configuración de GPO ubicada en el nivel de dominio se aplicará a todos los objetos de usuario y equipo dentro del dominio.
- ▶ Unidades organizativas (OU). Es el nivel de configuración más granular. Si la estructura de OU está bien planificada, resultará fácil hacer asignaciones lógicas de GPO a departamentos o unidades de negocio.

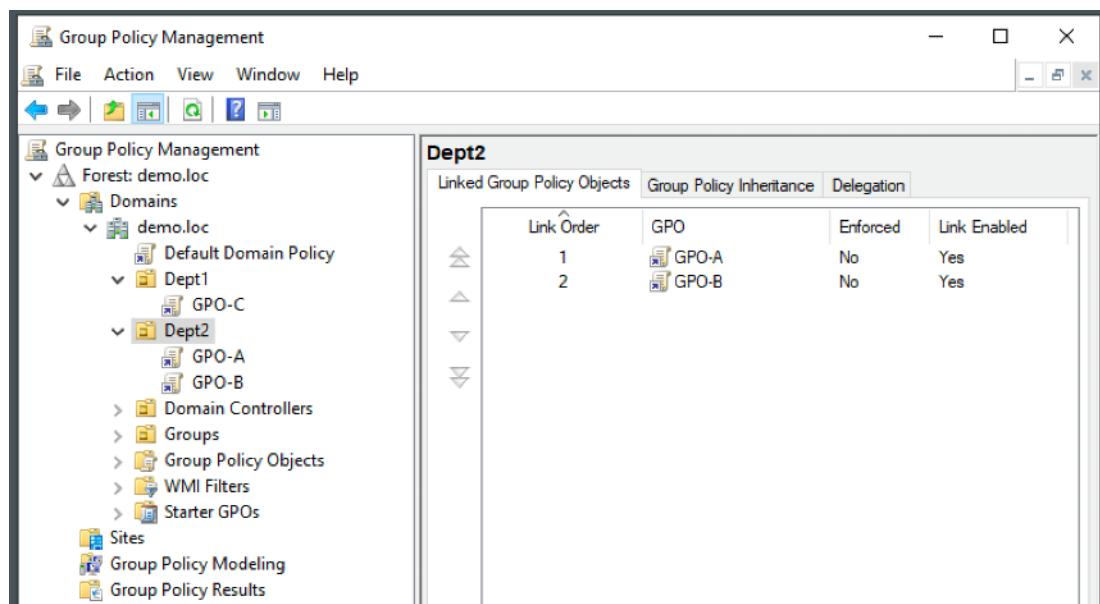


Figura 3. GPO a nivel de dominio y de unidad organizativa. Fuente: elaboración propia.

La Figura 3 muestra la consola de administración de directivas de grupo. La Default Domain Policy está enlazada a nivel de dominio, la GPO-C está enlazada en la OU Dept1 y las GPO-A y GPO-B están enlazadas en la OU Dept2.

Herencia de GPO

En la mayoría de los casos, la configuración de las directivas de grupo es **acumulativa**. Por ejemplo, un GPO a nivel de dominio podría especificar que todos los usuarios dentro del dominio deben cambiar su contraseña cada sesenta días, y un GPO en el nivel de OU podría especificar el fondo de escritorio predeterminado para todos los usuarios y equipos dentro de esa OU. En este caso, se aplican ambas configuraciones, por lo que los usuarios dentro de la unidad organizativa se ven obligados a cambiar su contraseña cada sesenta días y tienen la configuración predeterminada de escritorio.

¿Qué sucede si hay un **conflicto en la configuración**? Por ejemplo, en un escenario donde un GPO en el nivel del sitio especifica que los usuarios deben usar un fondo de pantalla rojo y otro GPO en el nivel de UO especifica que deben usar un fondo de pantalla verde. De manera predeterminada, la configuración en el nivel más específico (en este caso, la unidad organizativa que contiene el usuario) anula las de niveles más generales.

Las directivas de los GPO se aplican en el orden del listado anterior: primero las locales, luego las de sitio, las de dominio y, finalmente, las de OU, desde las OU superiores en el árbol a las más profundas. Prevalecerá la última configuración aplicada de una directiva. En el ejemplo anterior, los usuarios tendrán el fondo de pantalla verde.

Aunque el comportamiento predeterminado es que la configuración sea acumulativa y heredada, este comportamiento se puede modificar con **dos opciones**:

- ▶ Bloqueo de herencia. Especifica que las directivas de grupo para un objeto contenedor no se heredan de los contenedores superiores. Esto puede ser útil, por ejemplo, cuando una unidad organizativa secundaria requiere una configuración completamente diferente de una unidad organizativa principal.

- ▶ Forzar herencia de directivas. Cuando se activa esta opción en un GPO, se garantiza que todos los objetos de nivel inferior hereden estas configuraciones. En algunos casos, los administradores desean asegurarse de que la herencia de la directiva de grupo no esté bloqueada en otros niveles. Por ejemplo, se puede aplicar en una política corporativa de contraseña para que los administradores de OU inferiores no la bloquen (hay que recordar que las OU permiten delegar la administración de objetos, por lo que los administradores de una OU no tienen por qué ser los mimos que se encargan del dominio).

El último concepto que hay que considerar en la aplicación de GPO es que, si existe un conflicto entre la directiva aplicada a un equipo y a un usuario, prevalece la **configuración del usuario**. Esto es relevante, porque los objetos de usuario y de equipo no tienen por qué pertenecer a la misma OU y, por tanto, pueden recibir configuraciones diferentes. La configuración del usuario es más específica y permite a los administradores realizar cambios para usuarios individuales, independientemente del equipo en el que inician sesión.

9.3. PowerShell remota

PowerShell se puede usar como una herramienta de **administración remota**. Este capítulo muestra cómo hacerlo: en primer lugar, se explica cómo configurar un servidor para que acepte conexiones remotas de PowerShell y, a continuación, cómo conectarse al mismo para obtener información y realizar cambios.

Preparar el servidor remoto

Solo hay un par de elementos que deben ejecutarse y habilitarse en los servidores remotos para que acepten sesiones de PowerShell desde una máquina diferente. La comunicación remota de PowerShell está habilitada de manera predeterminada en equipos Windows Server 2012 y superiores y es posible que no sea necesario seguir

estos pasos. Sin embargo, si la funcionalidad ha sido deshabilitada manualmente o con una política de dominio, o si se intenta acceder a un equipo más antiguo, estos son los elementos para tener en cuenta:

- ▶ El servicio WinRM: el servicio WinRM es parte de la administración remota de Windows Server. Simplemente hay que asegurarse de que este servicio se esté ejecutando. Esto se puede verificar desde la consola de services.msc (es decir, la consola Servicios de MMC) o con el comando Get-Service WinRM.
- ▶ Enable-PSRemoting -Force: este comando debe ser ejecutado en cada servidor que vaya a aceptar conexiones remotas. Será necesaria una conexión por RDP para ejecutarlo, pero también se puede hacer preparando una imagen maestra con este comando. La ejecución de Enable-PSRemoting intenta iniciar el servicio WinRM (lo conseguirá si está parado, pero fallará si el servicio está deshabilitado), configura el sistema para aceptar conexiones remotas y crear una regla de *firewall* en el sistema para permitir este tráfico.
- ▶ Habilitar conexiones desde otros dominios o grupos de trabajo: si tanto el servidor administrado como el equipo que inicia la conexión están en el mismo dominio, como suele ser el caso en un entorno corporativo, entonces la autenticación entre máquinas es fácil de lograr, porque confían automáticamente entre sí. Sin embargo, si ambos equipos están en dominios diferentes, que no tienen una relación de confianza entre ellos, o si al menos uno de ellos pertenece a un grupo de trabajo (es decir, no es miembro de un dominio), entonces es necesario configurar el equipo manualmente, para que confíe en el equipo remoto que se va a conectar. Por ejemplo, si el equipo desde el que se inicia la conexión tiene win-admin como nombre de *host*, el comando necesario sería el siguiente:

```
Set-Item wsman:\localhost\client\Trustedhosts win-admin
```

Conexión al servidor remoto

Hay dos opciones a la hora de usar PowerShell de forma remota: por un lado, se pueden ejecutar cada comando en el sistema remoto de manera individual y, por otro, se puede abrir una sesión de PowerShell interactiva, de la misma manera que se puede iniciar una sesión interactiva de SSH en Linux.

La primera opción pasa por usar el parámetro `-ComputerName`. Muchos de los *cmdlets* disponibles en PowerShell, en particular los que comienzan con `Get-`, se pueden usar con el parámetro `-ComputerName`. Esto especifica que el comando en cuestión debe ejecutarse en el sistema remoto que especifique este parámetro. Por ejemplo, para comprobar si el servicio del programador de tareas está arrancado en el servidor `server1`, habría que ejecutar el siguiente comando:

```
Get-Service schedule -ComputerName server1
```

El parámetro acepta más de un nombre de *host*, por lo que, en caso de especificar varios, la salida mostrará la ejecución del comando en cada uno de los equipos.

Por otro lado, a veces es más cómodo usar `Enter-PSSession` e iniciar una sesión interactiva en la que ejecutar los *cmdlets*. Al ejecutar `Enter-PSSession`, la línea de comandos pasa a ser la *shell* del equipo remoto. Los *cmdlets* se ejecutarán en esa *shell*, que tiene su propio entorno y sus propias variables. Para arrancar una consola en el servidor `server1`, habría que ejecutar:

```
Enter-PSSession -ComputerName server1
```

En ambos casos, puede ocurrir que la cuenta de usuario del equipo sea diferente de la cuenta del equipo remoto. En ese caso, el parámetro `-Credential` sirve para indicar la cuenta de usuario. PowerShell abrirá un cuadro de diálogo para solicitar la contraseña.

```
Enter-PSSession -ComputerName server1 -Credential USERNAME
```

9.4. Herramientas de sistema

Este capítulo repasa algunas de las herramientas de mantenimiento y diagnóstico que un administrador usará en su día a día. Aunque no sean tan automatizables como PowerShell, porque son herramientas gráficas, son esenciales para facilitar la vida de cualquier administrador.

Visor de eventos

El Visor de eventos, o Event Viewer, es el visor de *logs* de Windows (Krause, 2019). Las aplicaciones pueden escribir sus *logs* en un fichero de texto, pero Windows ofrece un motor nativo para ello. Los eventos están categorizados en función de su origen (ver Figura 4), por lo que encontrar un evento concreto requiere revisar varias carpetas. El Visor de eventos se basa en la consola MMC, por lo que se puede acceder al Visor desde otro equipo sin necesidad de iniciar una sesión por RDP.

Los eventos precedidos por un icono azul son informativos, los amarillos indican una advertencia y los rojos son errores. De alguna manera, equivalen a los niveles de los tradicionales de INFO, WARNING y ERROR.

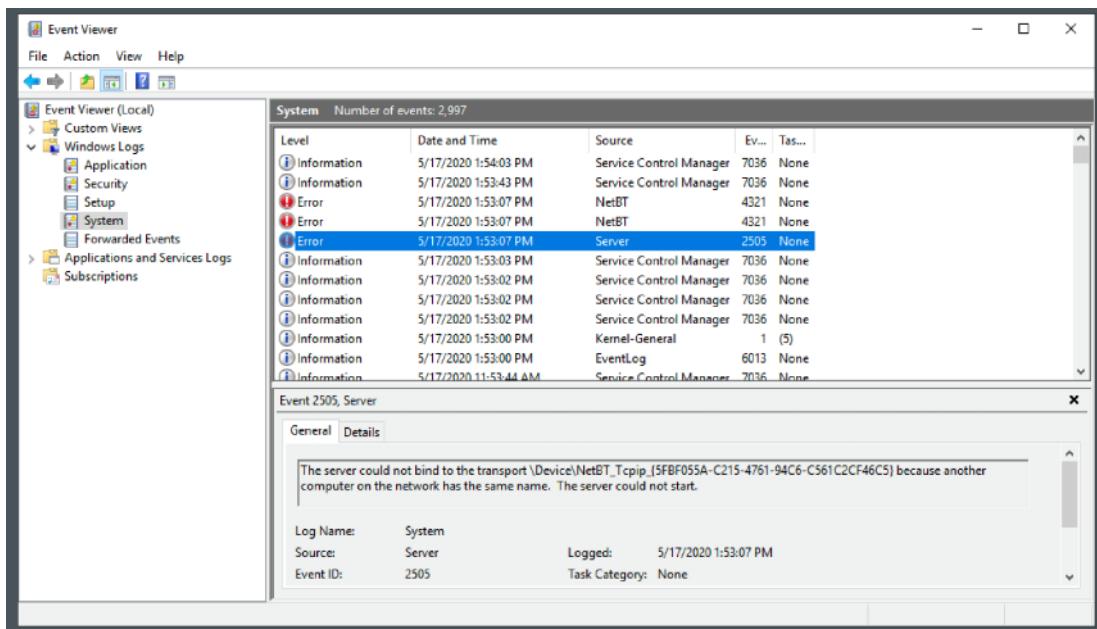


Figura 4. Visor de eventos. Fuente: elaboración propia.

Administrador de tareas

El Administrador de tarea, o Task Manager (Krause, 2019), es una herramienta que ha existido en todos los sistemas operativos Windows desde los primeros días de la interfaz gráfica, pero ha evolucionado bastante a lo largo de los años. Se invoca típicamente presionando **Ctrl + Alt + Supr** en el teclado y luego haciendo clic en Administrador de tareas o haciendo clic derecho en la barra de tareas y luego seleccionando Administrador de tareas. También se puede iniciar con la combinación de teclas **Ctrl + Shift + Esc** o escribiendo **taskmgr** en el cuadro de diálogo Ejecutar.

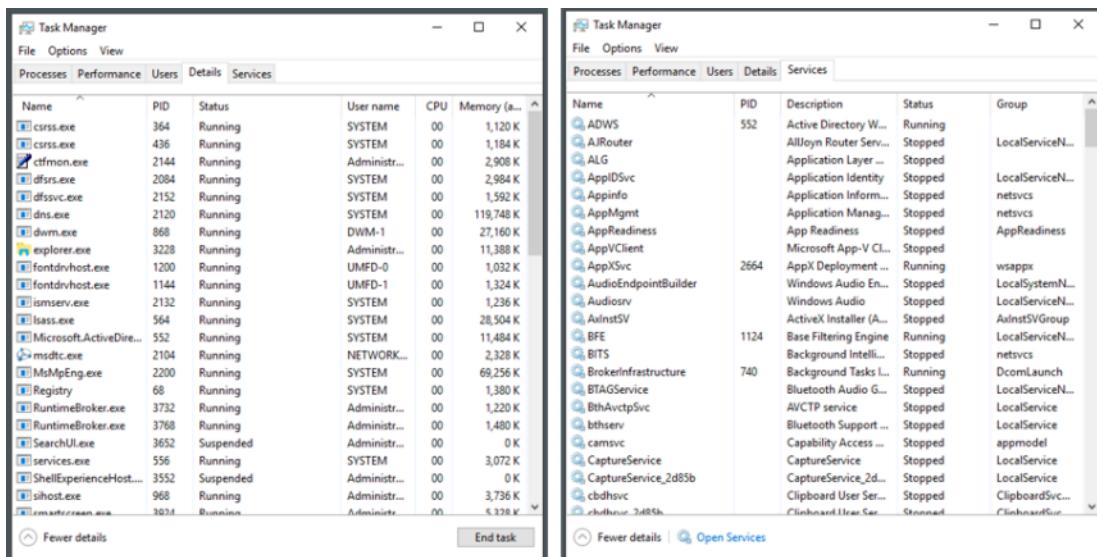


Figura 5. Administrador de tareas. Fuente: elaboración propia.

Las pestañas más relevantes para administrar procesos son **Detalles y Servicios**. La primera muestra qué aplicaciones se están ejecutando actualmente en el sistema. Se puede obtener información como el PID, el usuario que la ha iniciado, la llamada de línea de comandos, el uso de CPU y memoria, etc. Es, a grandes rasgos, un sustituto del comando top de Linux. Además, es posible detener la ejecución de un proceso con el menú contextual.

Algunos de los procesos pertenecerán a servicios de Windows. La pestaña Servicios muestra el estado de estos y permite arrancarlos, pararlos, etc. No es un sustituto completo de la consola Servicios de MMC, porque no se pueden ver y editar los detalles de cada servicio.

Programador de tareas

El Programador de tareas (Perrott, 2019), o Task Scheduler, permite programar tareas administrativas para que se **ejecuten automáticamente**. Muchos de los componentes de Windows tienen sus propias tareas listas para usar. Es, en cierta medida, equivalente a cron. Al igual que el Visor de eventos, el Programador de tareas es un *snapshot* de la consola MMC.

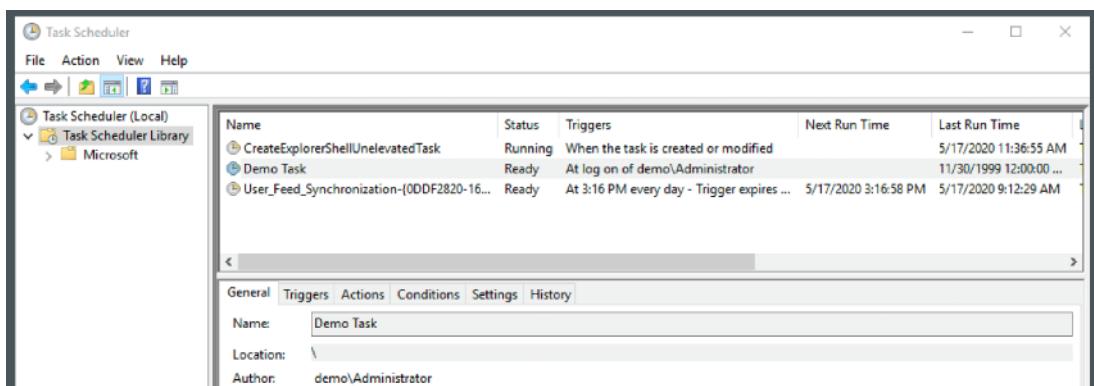


Figura 6. Programador de tareas. Fuente: elaboración propia.

Las tareas pueden ser llamadas a ejecutables o *scripts*, por lo que se puede programar cualquier tarea automatizable. Los desencadenantes, o *triggers*, pueden ser desde una hora concreta hasta un inicio de sesión, el arranque del equipo, un evento, un bloqueo de pantalla, etc.

Sysinternals

La web de Sysinternals, nacida en 1996 como **Ntinternals**, ofrece multitud de herramientas de diagnóstico para Windows. Vienen a cubrir la necesidad de muchos administradores, de tener herramientas potentes para tareas de mantenimiento que no son posibles, o al menos no son fáciles, en un sistema Windows recién instalado. Por ejemplo, **PsExec** es una herramienta de línea de comandos que permite ejecutar acciones y *scripts* en máquinas remotas. Actualmente, esto es relativamente fácil gracias a PowerShell, pero PsExec ha estado disponible mucho antes de la primera versión de esta consola.

Otra de las herramientas destacadas es **Process Explorer**. Viene a ser un sustituto del administrador de tareas: ofrece mucha más información de cada proceso y puede mostrar los procesos en orden jerárquico (ver Figura 7).

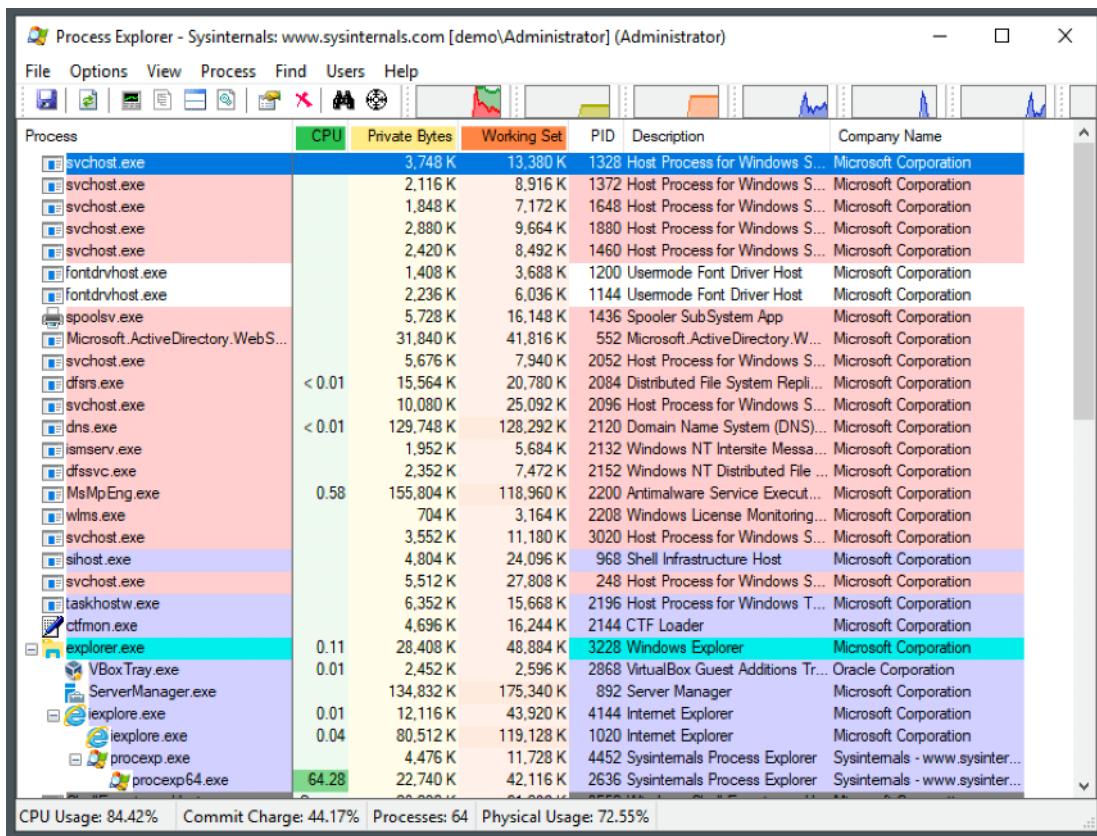


Figura 7. Process Explorer. Fuente: elaboración propia.

Aunque su objetivo puede parecer superfluo, la herramienta **BgInfo** está instalada en numerosos servidores. BgInfo se ejecuta en cada inicio de sesión y personaliza el fondo de pantalla con un bloque de texto. Se usa típicamente para mostrar información básica del equipo, como nombre de *host*, espacio libre en disco, dirección IP, etc. La Figura 8 muestra la pantalla de configuración de BgInfo y el fondo de escritorio que se obtiene tras aplicar la configuración.

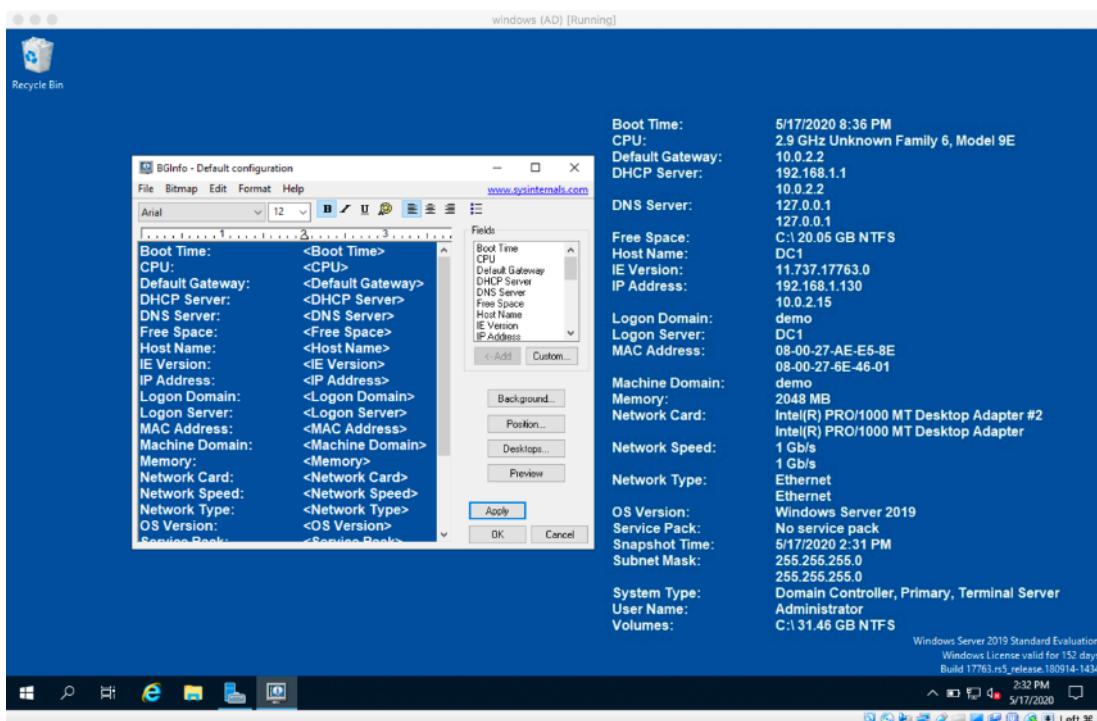


Figura 8. BgInfo. Fuente: elaboración propia.

TCPview es una alternativa gráfica a netstat. Muestra información detallada de los puertos abiertos y las conexiones establecidas.

Process	PID	Protocol	Local Address	Local Port	Remote Address	Remote Port
[System Proc...]	0	TCPV6	dc1.demo.loc	52482	dc1.demo.loc	47001
[System Proc...]	0	TCPV6	[fe80::0:9ceb:a...]	52484	[fe80::0:9ceb:a...]	epmap
[System Proc...]	0	TCP	dc1.demo.loc	52478	40.115.3.210	https
dftrs.exe	2084	TCPV6	[fe80::0:9ceb:a...]	49719	[fe80::0:9ceb:a...]	ldap
dftrs.exe	2084	TCPV6	[fe80::0:9ceb:a...]	49721	[fe80::0:9ceb:a...]	49667
dftrs.exe	2084	TCPV6	[fe80::0:9ceb:a...]	49722	[fe80::0:9ceb:a...]	ldap
dfssvc.exe	2152	TCPV6	[fe80::0:9ceb:a...]	49689	[fe80::0:9ceb:a...]	49667
dns.exe	2120	TCPV6	dc1.demo.loc	49682	dc1.demo.loc	ldap
dns.exe	2120	TCPV6	[fe80::0:9ceb:a...]	49685	[fe80::0:9ceb:a...]	ldap
ismserv.exe	2132	TCPV6	dc1.demo.loc	49674	dc1.demo.loc	ldap
ismserv.exe	2132	TCPV6	dc1.demo.loc	49675	dc1.demo.loc	ldap
lsass.exe	564	TCPV6	dc1.demo.loc	ldap	dc1.demo.loc	49674
lsass.exe	564	TCPV6	dc1.demo.loc	ldap	dc1.demo.loc	49675
lsass.exe	564	TCPV6	dc1.demo.loc	ldap	dc1.demo.loc	49682
lsass.exe	564	TCPV6	[fe80::0:9ceb:a...]	ldap	[fe80::0:9ceb:a...]	49685
lsass.exe	564	TCPV6	[fe80::0:9ceb:a...]	ldap	[fe80::0:9ceb:a...]	49719
lsass.exe	564	TCPV6	[fe80::0:9ceb:a...]	ldap	[fe80::0:9ceb:a...]	49722
lsass.exe	564	TCPV6	[fe80::0:9ceb:a...]	49667	[fe80::0:9ceb:a...]	49689
lsass.exe	564	TCPV6	[fe80::0:9ceb:a...]	49667	[fe80::0:9ceb:a...]	49721
lsass.exe	564	TCPV6	[fe80::0:9ceb:a...]	49667	[fe80::0:9ceb:a...]	49777
lsass.exe	564	TCPV6	[fe80::0:9ceb:a...]	49777	[fe80::0:9ceb:a...]	49667
svchost.exe	1168	TCP	dc1.demo.loc	52181	40.67.254.36	https
svchost.exe	1168	TCP	dc1.demo.loc	52359	40.67.254.36	https
svchost.exe	776	TCP	dc1.demo.loc	52490	192.168.1.133	epmap
svchost.exe	776	TCP	dc1.demo.loc	52491	192.168.1.133	epmap

At the bottom, it shows network statistics: Endpoints: 27, Established: 38, Listening: 0, Time Wait: 3, Close Wait: 0.

Figura 9. TCPview. Fuente: elaboración propia.

9.5. Referencias bibliográficas

Krause, J. (2019). *Mastering Windows Server 2019: The Complete Guide for IT Professionals to Install and Manage Windows Server 2019 and Deploy New Capabilities* (2.ª ed.). Packt Publishing.

Microsoft. (2021, junio 22). *Windows Sysinternals*.

<https://docs.microsoft.com/en-us/sysinternals/>

Panek, W. (2018). *MCSA Windows Server 2016 Complete Study Guide*. Sybex.

Perrott, S. (2019). *Windows Server 2019 & PowerShell All-in-One For Dummies*. Wiley.

Administración de Sistemas en la Cloud

Instalación y administración en *cloud*

Índice

Esquema	3
Ideas clave	4
10.1. Introducción y objetivos	4
10.2. Despliegue de aplicaciones en la nube	4
10.3. Sysprep	9
10.4. Creación de imágenes en AWS	14
10.5. Creación de imágenes Windows en Azure	18
10.6. Referencias bibliográficas	21

Esquema

INSTALACIÓN Y ADMINISTRACIÓN EN CLOUD

Despliegue de aplicaciones en la nube

- ▶ Despliegue tradicional: actualización incremental, servidores «mascota».
- ▶ Despliegue basado en instancias: imagen base e instalación de la aplicación de cero.
- ▶ Despliegue basado en imágenes: nueva imagen en cada versión.

Sysprep

- ▶ Herramienta de generalización de servidores Windows.
- ▶ Elimina elementos específicos para poder reusar una instalación de sistema operativo.

10.1. Introducción y objetivos

Los comandos y herramientas de administración y automatización no son, salvo en algunos casos, específicos de entornos de nube. Al fin y al cabo, **los sistemas operativos no han cambiado drásticamente** por el hecho de estar disponibles en proveedores de nube. Los administradores pueden, por tanto, seguir trabajando con los sistemas con muchas de las herramientas a las que están acostumbrados.

El apartado de despliegue de servidores, sin embargo, ha cambiado mucho respecto a la instalación de servidores físicos. Las ideas generales siguen en pie (actualización paulatina de aplicaciones, generalización de servidores, etc.), pero las **herramientas se han adaptado** a las características de la nube para acelerar los despliegues.

Los **objetivos** que se pretenden conseguir en este tema son:

- ▶ Entender los diferentes enfoques de actualización de aplicaciones en entornos de nube.
- ▶ Conocer el concepto de generalización aplicado a imágenes.
- ▶ Tomar contacto con la creación de imágenes en AWS y Azure.

10.2. Despliegue de aplicaciones en la nube

Los pasos para actualizar una aplicación propia en ejecución se podrían generalizar de la siguiente manera:

- ▶ Un desarrollador escribe un código para implementar una nueva característica.

- ▶ Los archivos modificados se incorporan a un sistema de control de versiones tras pasar las pruebas.
- ▶ Dependiendo del lenguaje de programación que se utilice, puede ser necesario compilar los archivos fuente para producir un binario ejecutable.
- ▶ Los archivos fuente modificados (o los ejecutables compilados) están disponibles para las instancias en ejecución. Las instancias pueden extraer los archivos directamente del sistema de control de versiones o, tal vez, usar un repositorio local para paquetes de APT, Yum, Ruby, Python u otro tipo de gestor de paquetes.
- ▶ Una vez los archivos modificados están en las instancias, los servicios en ejecución se reinician para empezar a usar el nuevo código o los nuevos archivos de configuración.

Esto es una descripción general de alto nivel y algunas aplicaciones pueden requerir pasos adicionales.

En un entorno tradicional, con servidores físicos, las tareas de desplegar servidores y desplegar o actualizar aplicaciones eran acciones claramente separadas: un servidor con un rol concreto se desplegaba una vez y la aplicación se actualizaba tantas veces como fuera necesario.

En un entorno virtualizado, ya sea de nube pública o privada, hay **dos enfoques** habituales para estas actualizaciones (Lucifredi y Ryan, 2018):

- ▶ Despliegue basado en instancias: cada instancia se actualiza individualmente en el momento del despliegue de la aplicación.
- ▶ Despliegue basado en imágenes: se crea una nueva imagen cada vez que se lanza una versión de producción, a partir de la cual se crean nuevas instancias y se destruyen las antiguas.

Por supuesto, el proceso tradicional se puede replicar en un entorno de nube, pero estos dos enfoques son los que tienen sentido en un entorno DevOps.

Despliegue basado en instancias

En este modelo de despliegue, todos los roles de servidor se basan en una única **imagen maestra** (o unas pocas, en caso de necesitar diferentes sistemas operativos o distribuciones para diferentes roles). La configuración del rol ocurre durante el arranque de la instancia mediante la ejecución de un *script* a partir de algún mecanismo concreto: *user data* en AWS, extensiones de [máquina virtual](#) de Azure o herramientas con Ansible o Puppet.

Una posible actualización de una aplicación de la versión 1.0 a la versión 1.1 con este modelo seguiría estos pasos:

- ▶ Una vez completados los pasos generales, los binarios o los paquetes de actualización, se etiquetan con el número de versión 1.1.
- ▶ Se modifican los *scripts* de despliegue para hacer referencia a los paquetes de la versión 1.1.
- ▶ Todas las instancias desplegadas a partir de este momento ejecutarían los *scripts* para instalar la versión 1.1, pero las instancias ya desplegadas siguen en la versión 1.0. En un grupo de autoescalado, en el que pueden aparecer instancias nuevas en cualquier momento, se podría llegar a una situación con instancias en ambas versiones simultáneamente.
- ▶ Idealmente, las instancias en la versión 1.0 se destruyen y son sustituidas por instancias nuevas, que tendrán la versión 1.1. Esta sustitución ocurre paulatinamente en un *rolling update*. Para que una sustitución paulatina pueda ocurrir, ambas versiones deben poder convivir, incluso aunque sea momentáneamente. De lo contrario, pueden aparecer problemas con los esquemas de las bases de datos u otras configuraciones. Si no pueden convivir, es probable que sea necesaria una actualización en bloque en la que habrá un intervalo sin servicio.

En ambos casos, es habitual que las instancias estén detrás de un balanceador de carga que solo dirige tráfico a las instancias que pasan una llamada de prueba o *healthcheck* (que puede ser, por ejemplo, una petición REST a una ruta diseñada expresamente para validar que la instancia funciona bien). No todas las instancias tienen por qué alojar un servidor web: un servicio que se alimente de una cola de mensajes puede evitar la llamada de *healthcheck*, ya que el propio servicio no recogerá mensajes hasta que no esté listo para hacerlo.

Se podría pensar que se podrían aplicar los *scripts* de la versión 1.1 en las instancias con la versión 1.0 para evitar un despliegue completo. Este enfoque es realmente más parecido al tradicional, en el que se aplican parches sobre un servidor indefinidamente.

Despliegue basado en imágenes

La creación de nuevas imágenes proporciona una forma más limpia de ejecutar actualizaciones. Estas imágenes, o plantillas, consisten en una máquina virtual con una configuración y unas aplicaciones concretas, que se apaga y se «congela», en el sentido de que no puede ser arrancada. Esta imagen mantiene la configuración del *hardware* virtual y los discos de la máquina sin modificaciones desde el apagado. Las instancias se despliegan «clonando» esta imagen, es decir, creando una nueva máquina virtual con el mismo *hardware* virtual y con un disco que es una copia del disco de la imagen.

Los **pasos** para una actualización de 1.0 a 1.1 son los siguientes:

- ▶ Al igual que con el enfoque anterior, una vez completados los pasos generales, se generan los paquetes de instalación con el número de versión 1.1.
- ▶ Se arranca una instancia con una instalación básica del sistema operativo, que podría hacerse con los últimos parches de seguridad. Este sistema operativo suele

ser el mismo que el de la imagen de la versión 1.0, pero no es estrictamente necesario.

- ▶ Se instala la aplicación, versión 1.1, desde cero. Los *scripts* de instalación serán parecidos a los del modelo basado en instancias, pero no tienen que tratar con actualizaciones de la aplicación, solo con la instalación inicial, por lo que pueden ser más sencillos.
- ▶ Una vez completada la instalación, se «generaliza» la imagen y se apaga la máquina virtual (en algunos entornos este proceso es automático).
- ▶ Dependiendo del entorno de nube, la imagen se genera a partir de la máquina virtual, dejando la máquina intacta, o la propia máquina virtual se convierte en una plantilla. En ambos casos, la imagen es inmutable y no se puede arrancar, salvo mediante un clonado.
- ▶ Las instancias en ejecución, basadas en la imagen de la versión 1.0, se sustituyen por instancias nuevas con la versión 1.1. Al igual que en el modelo anterior, se puede seguir un proceso de *rolling update* o una actualización en bloque con parada del servicio. La problemática de convivencia de varias versiones existe también en este modelo de despliegue.

La actualización basada en imágenes es más estable, porque no hay actualizaciones incrementales, sino instalaciones de cero. Como contrapartida, generar una imagen nueva, aun cuando el proceso esté automatizado, lleva más tiempo que cambiar un *script* de despliegue. En cualquier caso, el despliegue basado en imágenes puede seguir haciendo uso de *scripts* de arranque (es decir, *user data*, extensiones, etc.) para finalizar la configuración de la aplicación, en caso de que sea necesario.

Inmutabilidad de la aplicación

El despliegue tradicional sufre de un hecho conocido como **derivado de la configuración**. Esta situación ocurre en el escenario de servidores «mascota», en el que los administradores tratan con cariño de mantener los servidores con vida a toda costa. Cualquier diferencia en la aplicación de una configuración puede dar lugar a horas de investigación para resolver un problema. Cuando los administradores trabajan con servidores «ganado», todas las instancias están en un estado conocido, ya que todas parten de un mismo punto de partida, sea un *script* de instalación o una imagen maestra. El corolario es que, una vez que se despliega un servidor, nunca debe modificarse, solo debe reemplazarse. Esto se conoce como el patrón de diseño del servidor inmutable.

La inmutabilidad de la aplicación se logra automatizando las operaciones realizadas en las instancias. Estas operaciones se limitan, exclusivamente, al aprovisionamiento, reemplazo y desaprovisionamiento. Si se consigue implementar este patrón por completo, la aplicación se puede desplegar de manera predecible, los cambios se pueden revertir con la misma facilidad y su estado completo, incluida la configuración de seguridad, se entiende claramente y se puede reproducir de manera exacta en cualquier punto del ciclo de vida de la instancia.

10.3. Sysprep

Una de las principales ventajas de los entornos de nube es la capacidad de desplegar nuevos servidores rápidamente. Si bien se podría seguir el procedimiento tradicional de crear una máquina virtual, conectar una imagen ISO, instalar el sistema operativo, instalar las aplicaciones, etc., las nubes privadas y públicas ofrecen funcionalidades que aceleran este proceso. El *user data* de AWS es un ejemplo, como lo son las imágenes personalizadas.

Windows está preparado para estas tecnologías. Este capítulo trata sobre una herramienta que facilita la automatización del despliegue en cualquier entorno de nube pública o privada. Está integrada en todas las versiones de Windows, por lo que puede llevar a cabo este mismo proceso en cualquier máquina Windows actual, ya sea un cliente o un servidor.

Sysprep (Krause, 2019) es una herramienta que prepara el sistema para el clonado, aplicando un proceso llamado **generalización**. Su nombre oficial es Herramienta de preparación del sistema de Microsoft. En resumen, permite crear una imagen maestra de un servidor que puede reutilizarse tantas veces como sea necesario para desplegar servidores adicionales.

Una **ventaja** clave del uso de Sysprep es que es posible añadir configuraciones personalizadas en el servidor maestro e instalar elementos, como las actualizaciones de Windows, y todas estas configuraciones y parches existirán en la imagen maestra. El uso de Sysprep ahorra tiempo, al no tener que recorrer el proceso de instalación del sistema operativo, pero ahorra aún más al no tener que esperar a que las actualizaciones de Windows desplieguen todos los parches actuales en cada nuevo sistema.

¿Es Sysprep necesario? Se podría pensar que, para clonar un servidor maestro, simplemente se podría usar una herramienta de clonado de disco duro o, si se tratara de máquinas virtuales, simplemente se podría copiar y pegar el fichero de disco virtual para hacer una copia de su nuevo servidor. Esto puede funcionar sobre el papel, pero el principal problema es que la nueva imagen sería una réplica exacta de la original: el nombre del *host* sería el mismo y, lo que es más importante, cierta información de identificación central en Windows, como el número de identificador de seguridad (SID) del sistema operativo, sería exactamente el mismo. Si activara tanto el servidor maestro original como un servidor nuevo basado en esta réplica exacta, aparecerían conflictos y colisiones en la red, ya que estos dos servidores luchan por su derecho a ser el único servidor con ese nombre único y SID.

Este problema es especialmente relevante en los entornos de Directorio Activo, donde es aún más importante que cada equipo miembro del dominio red tenga un SID único. Sysprep corrige todos estos problemas inherentes al proceso de duplicación del sistema al generar identificadores únicos aleatorios durante el primer arranque de los servidores clonados.

Las **fases** para preparar una imagen maestra con Sysprep son las siguientes:

- ▶ Instalación del sistema operativo en un nuevo servidor.
- ▶ Configuración y actualización.
- ▶ Ejecución de Sysprep y apagado del servidor.
- ▶ Creación de imagen maestra del disco duro.
- ▶ Clonado del disco para desplegar nuevos servidores.

Instalación, configuración y actualización

Para la instalación y configuración del equipo, se puede seguir el proceso habitual, pero en general no es recomendable instalar roles de Windows, porque, en función del rol, el proceso de Sysprep puede causar problema con la configuración. Esto se debe a que una instalación de roles realiza numerosos cambios en el sistema operativo y algunos de los roles bloquean el nombre de *host* particular del servidor maestro, aun cuando el nombre será diferente en el clon. Las personalizaciones que consisten en crear archivos y carpetas, arrancar o detener servicios o cambiar configuraciones del registro suelen ser seguras. La instalación de parches de seguridad es fundamental, ya que es una de las fases que más tiempo consume tras el despliegue.

Ejecutar Sysprep

Ahora que el servidor maestro está preparado, es hora de ejecutar Sysprep. El ejecutable se encuentra en C:\Windows\System32\Sysprep (ver Figura 1).

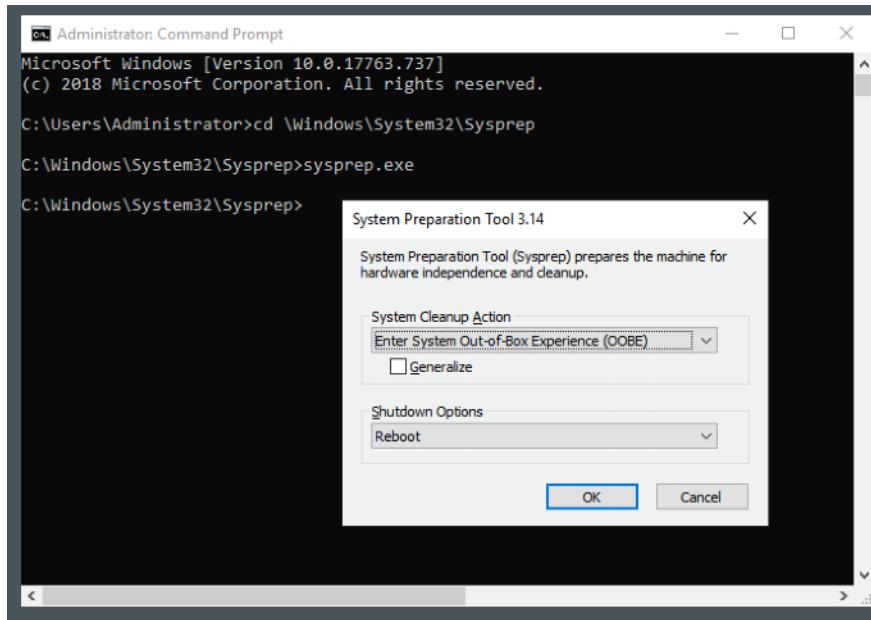


Figura 1. Ejecución gráfica de Sysprep. Fuente: elaboración propia.

Sysprep también admite modificadores de línea de comandos para configurar las mismas opciones que el cuadro de diálogo gráfico:

- ▶ **/generalize:** Sysprep eliminará toda la información única del sistema (SID) de la instalación de Windows, haciendo que la imagen final sea utilizable en múltiples máquinas, ya que cada nuevo servidor creado a partir de la imagen obtendrá un SID nuevo único.
- ▶ **/audit:** esto reinicia la máquina en un modo de auditoría especial, donde es posible agregar controladores adicionales a Windows antes de completar el proceso.
- ▶ **/oobe:** con este modificador activado, el servidor iniciará el asistente de mini configuración (el *out-of-box-experience*) la próxima vez que arranque Windows, es decir, en el primer arranque de cada servidor clonado.
- ▶ **/quiet:** que se ejecute Sysprep sin mensajes de estado en la pantalla.
- ▶ **/shutdown:** esto apaga el sistema cuando Sysprep finaliza.

Los dos que son más importantes para el propósito de crear una imagen maestra son los modificadores **/generalize** y **/shutdown**. El primero es fundamental, porque reemplaza toda la información de identificación única, la información SID, en las

nuevas copias de Windows de los servidores clonados. El modificador de apagado también es muy importante, ya que, en caso de un reinicio, el servidor generará un nuevo SID y habría que ejecutar Sysprep de nuevo. De no usarlo siempre, es posible apagar el servidor manualmente.

Para automatizar el arranque de los nuevos servidores lo más posible, el comando de Sysprep sería el siguiente:

```
sysprep.exe /generalize /shutdown
```

Después de iniciar este comando, Sysprep mostrará algunos mensajes y, finalmente, apagará el servidor.

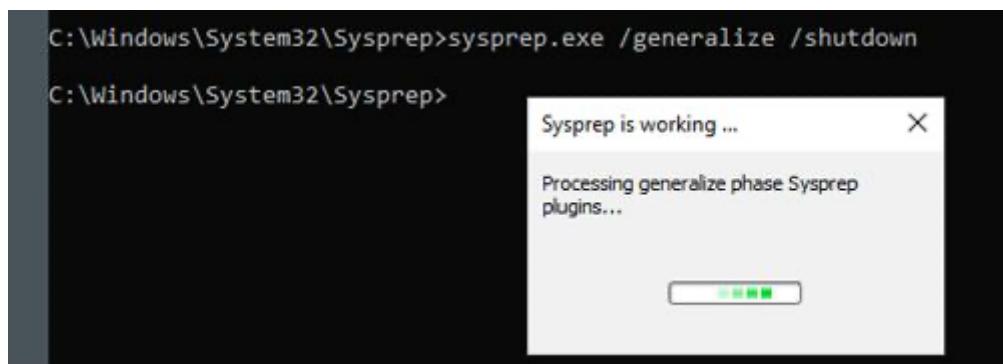


Figura 2. Sysprep preparando Windows. Fuente: elaboración propia.

Creación de la imagen maestra

Una vez el servidor está apagado, ha llegado el momento de crear la imagen maestra. En los servidores físicos tradicionales, había que arrancar desde un CD con herramientas que clonaban el contenido del disco a una unidad de red o a un disco duro externo. En un entorno virtual, la situación es más sencilla: basta con copiar el archivo del disco duro (por ejemplo, el fichero .vmx en vSphere o .vhdx en Hyper-V) o, lo que es más habitual, crear una plantilla a partir de la máquina virtual.

Las plantillas son similares a las máquinas virtuales, en cuanto a que consisten en una configuración de *hardware* virtual y uno o varios archivos de disco. Pero, en vez de

ofrecer la opción de arranque, el sistema de virtualización solo permite clonar plantillas en una máquina virtual nueva. De esta manera, los discos de la plantilla no se ven afectados por un arranque accidental que podría echar a perder el sellado del Sysprep.

Sysprep en la nube

Algunos proveedores de nube integran Sysprep en sus herramientas de creación de imágenes. Por ejemplo, [EC2Launch](#) o [EC2 Image Builder](#) en AWS son capaces de ejecutar Sysprep automáticamente. VMware Horizon soporta Sysprep, además de [QuickPrep](#), una variante propia que no aplica los mismos cambios que Sysprep, pero es útil en el despliegue de sistemas operativos de escritorio.

El cualquier caso, Sysprep es una herramienta general y, como tal, puede ejecutarse independientemente del proveedor de nube, incluso aunque este soporte Sysprep de manera integrada. Tanto [Azure](#) como AWS disponen de documentación sobre Sysprep que, a grandes rasgos, sigue el procedimiento descrito en este capítulo.

10.4. Creación de imágenes en AWS

Este capítulo muestra paso a paso cómo crear una [AMI](#), o Amazon *machine image*, a partir de una máquina virtual Linux en ejecución. El proceso en instancias Windows es algo más complejo, ya que requiere un proceso de generalización y no soporta Cloud-init directamente. En la sección A fondo hay un guía para crear imágenes Windows en AWS con EC2Launch.

A continuación, en el vídeo, *Despliegue de sistemas operativos en la nube*, se verán demostrados los pasos explicados a lo largo del tema.



Accede al vídeo

El primer paso es arrancar la imagen con la configuración necesaria. El tamaño, las opciones de redes y las etiquetas no formarán parte de la nueva AMI. La AMI inicial (ver Figura 3) sí que es relevante, ya que la nueva AMI será un incremento sobre la AMI base. El almacenamiento también es importante, ya que los discos EBS formarán parte de la AMI.

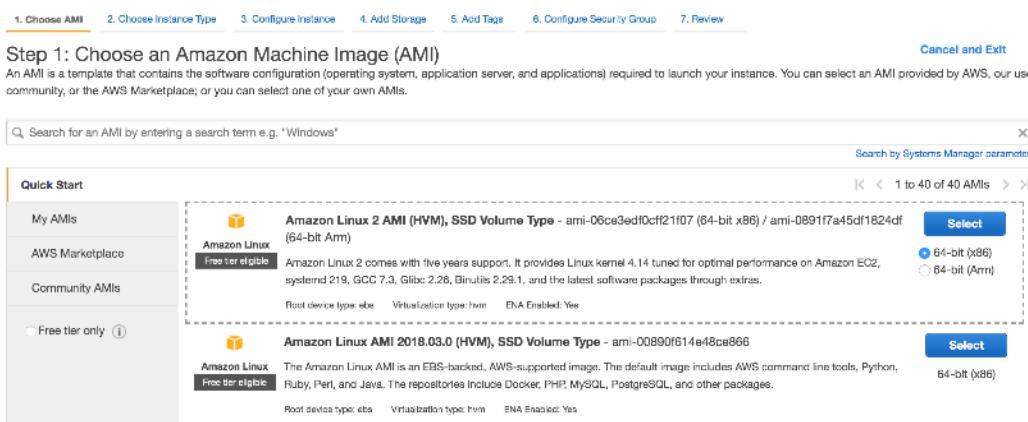


Figura 3. Selección de AMI base. Fuente: elaboración propia.

La instalación de aplicaciones y configuración es el siguiente paso. Se pueden crear usuarios por defecto, crear carpetas, añadir repositorios, etc. (ver Figura 4). En este paso, **no** es recomendable aplicar configuraciones dependientes de la instancia, como el nombre de *host* o configuraciones dependientes de la IP (por ejemplo, configurar un servidor para que escuche en una IP concreta), ya que esos valores cambiarán cuando arranquen las instancias basadas en esta imagen.

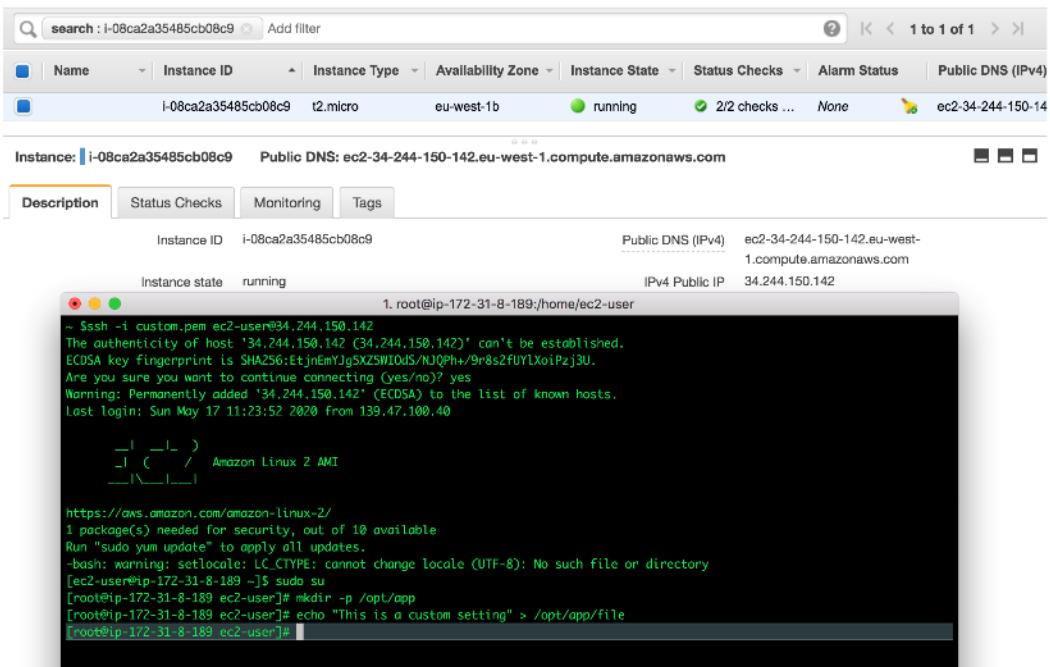


Figura 4. Conexión a la instancia, instalación y configuración de aplicaciones. Fuente: elaboración propia.

Cuando la instancia está lista, el siguiente paso es crear la imagen en el menú Actions > Image > Create Image (Figura 5). En el siguiente cuadro de diálogo, no hay más que introducir un nombre de imagen y una descripción. También permite añadir discos EBS adicionales, que estarán disponibles en todas las instancias arrancadas a partir de esta AMI.

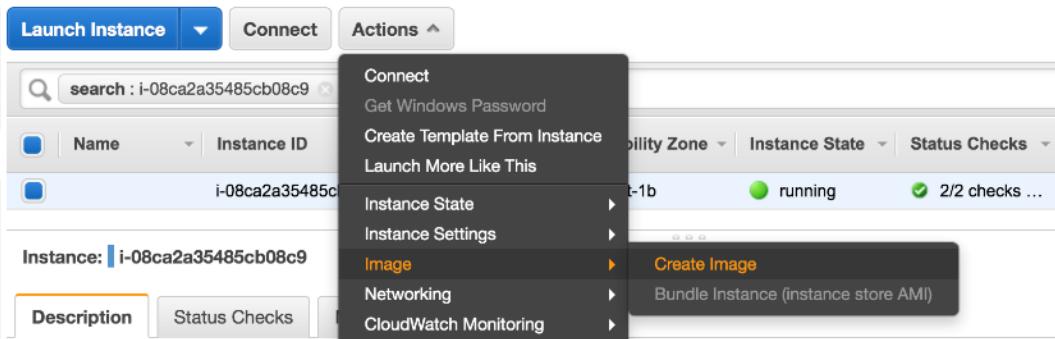


Figura 5. Menú de creación de instancia. Fuente: elaboración propia.

Como resultado de este proceso, aparecerán dos recursos: la propia AMI (ver Figura 6) y una captura de disco o *snapshot* (Figura 7). Esta captura de disco es equivalente a las imágenes de disco de servidores físicos o a los archivos de discos virtuales de otros entornos de virtualización.

Owned by me							
	Name	AMI Name	AMI ID	Source	Owner	Visibility	Status
	custom-ami		ami-05c5d35c7f6b1c528	014641795089/c...	014641795089	Private	pending

Image: ami-05c5d35c7f6b1c528		
Details	Permissions	Tags
AMI ID: ami-05c5d35c7f6b1c528 Owner: 014641795089 Status: pending Creation date: May 17, 2020 at 1:26:33 PM UTC+2 Architecture: x86_64	AMI Name: custom-ami Source: 014641795089/custom-ami State Reason: - Platform details: Linux/UNIX Usage operation: RunInstances	Edit

Figura 6. Listado de AMI privadas. Fuente: elaboración propia.

Create Snapshot Actions						
Owned By Me Filter by tags and attributes or search by keyword						
	Name	Snapshot ID	Size	Description	Status	Started
<input type="checkbox"/>	snap-08bebddc2b4...	8 GiB	Created by CreateImage(i-074e7041194cd1393) for ami-00706...	completed	May 17, 2020	8:45:42 AM
<input type="checkbox"/>	snap-0f43f143489...	8 GiB	Created by CreateImage(i-08ca2a35485cb08c9) for ami-05c5d...	completed	May 17, 2020	8:45:42 AM

Figura 7. Listado de capturas de disco (snapshots). Fuente: elaboración propia.

La nueva AMI estará disponible en la pestaña My AMIs en el asistente de despliegue de la nueva instancia. Si el despliegue se automatiza a través de un SDK o de la herramienta de línea de comandos awscli, solo es necesario usar el ID de la AMI, sin especificar si es pública o privada.

Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Search for an AMI by entering a search term e.g. "Windows"

My AMIs

custom-ami - ami-05c5d35c7f6b1c528

Custom Amazon Linux 2

Root device type: ebs Virtualization type: hvm Owner: 014641795089 ENA Enabled: Yes

Select

64-bit (x86)

Figura 8. AMI privada disponible para desplegar una instancia nueva. Fuente: elaboración propia.

```

~ $ssh -i custom.pem ec2-user@34.253.184.136
The authenticity of host '34.253.184.136 (34.253.184.136)' can't be established.
ECDSA key fingerprint is SHA256:EtjnEmYJg5XZ5W10d5/NJQPh+/9r8s2fUYlXoiPzj3U.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '34.253.184.136' (ECDSA) to the list of known hosts.
Last login: Sun May 17 11:25:09 2020 from 139.47.100.40

 _I_ _I_
 _I_ C_ / Amazon Linux 2 AMI
 ___\_\_\_I

https://aws.amazon.com/amazon-linux-2/
1 package(s) needed for security, out of 10 available
Run "sudo yum update" to apply all updates.
-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file or directory
[ec2-user@ip-172-31-12-170 ~]$ cat /opt/app/file
This is a custom setting
[ec2-user@ip-172-31-12-170 ~]$

```

Figura 9. Opciones personalizadas en la nueva instancia. Fuente: elaboración propia.

En AWS, las AMI son un recurso específico de una región. Es decir, esta AMI se podrá desplegar en la misma región en la que se creó. Para poder desplegarla en otras [regiones](#), es necesario copiarla primero.

10.5. Creación de imágenes Windows en Azure

Este capítulo muestra la creación de una imagen [Windows en Azure](#). En este caso, el proceso sí incluye el proceso de generalización.

Al igual que en AWS, el primer paso es desplegar la máquina virtual (Figura 10). De nuevo, las opciones como configuración de red, tamaño y etiquetas no son relevantes, ya que se podrán decidir en las futuras imágenes. La imagen inicial sí afectará a la imagen nueva.

Instance details

Virtual machine name *	base-vm
Region *	(Europe) France Central
Availability options	No infrastructure redundancy required
Image *	Windows Server 2019 Datacenter Browse all public and private images
Azure Spot instance	<input type="radio"/> Yes <input checked="" type="radio"/> No
Size *	Standard B2s 2 vcpus, 4 GiB memory (€36.94/month) Change size
Administrator account	
Username *	custom-admin
Password *	*****
Confirm password *	*****
Inbound port rules	
Select which virtual machine network ports are accessible from the public internet. You can specify more limited or granular network access on the Networking tab.	
Public inbound ports *	<input type="radio"/> None <input checked="" type="radio"/> Allow selected ports
Select inbound ports *	RDP (3389)

Figura 10. Creación de VM inicial. Fuente: elaboración propia.

Tras instalar las aplicaciones y aplicar las configuraciones necesarias, es necesario ejecutar Sysprep siguiendo los pasos indicados en el apartado *Despliegue de aplicaciones en la nube*.

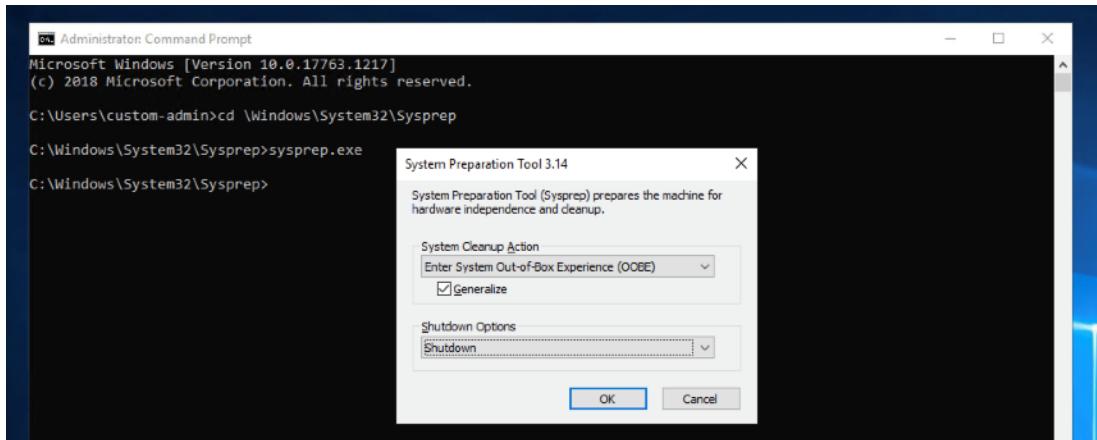


Figura 11. Generalización con Sysprep. Fuente: elaboración propia.

La ejecución de Sysprep debe terminar con un apagado de la VM. Una vez apagada, hay que pulsar el botón de Capture (Figura 12).

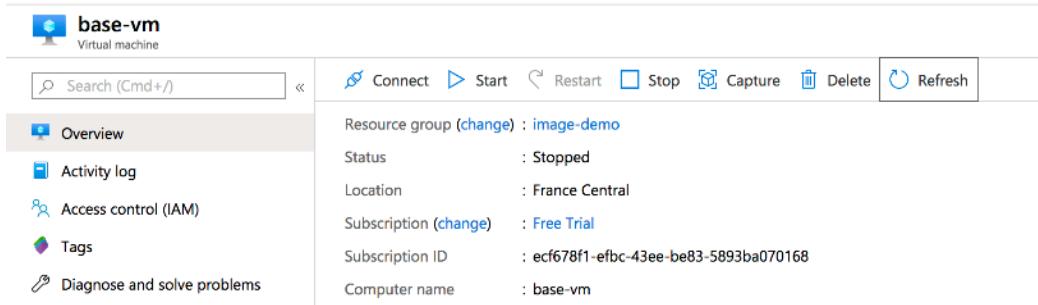


Figura 12. Botón Capture en la VM apagada. Fuente: elaboración propia.

A diferencia de AWS, donde la instancia convertida en AMI sigue funcionando normalmente, la creación de la imagen en Azure destruye la VM original, tal como indica el cuadro de diálogo de la Figura 13.

A screenshot of the 'Create image' dialog box in the Azure portal. It shows fields for Name (base-image), Resource group (image-demo), and options like automatic deletion and zone resiliency. A warning message at the bottom states that capturing the image will make the VM unusable.

Figura 13. Menú de creación de imagen a partir de la VM. Fuente: elaboración propia.

Una vez completado el proceso, la imagen estará disponible para su despliegue como VM. El botón Create VM de la Figura 14 iniciará el asistente de despliegue con esta imagen ya seleccionada.

The screenshot shows the AWS CloudFormation console. The top navigation bar says "All services > Images > base-image". On the left, there's a sidebar with options: Overview (selected), Activity log, Access control (IAM), Tags, Settings (Locks, Export template), and Support + troubleshooting. The main content area shows a stack named "base-image". It has fields for "NAME" (base-image), "SOURCE VIRTUAL MACHINE" (base-vm), and "OS DISK" (OS type Windows, Source blob URI empty). Below that, it says "DATA DISKS" with the note "This image doesn't contain any data disks." There are also "Create VM" and "Delete" buttons at the top right.

Figura 14. Imagen lista para ser desplegada. Fuente: elaboración propia.

10.6. Referencias bibliográficas

AWS. (s. f.). Configure EC2Launch. En *Configure a Windows instance using EC2Launch*.
<https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/ec2launch.html#ec2launch-config>

AWS. (s. f.). *Copy an AMI*.

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/CopyingAMIs.html>

AWS. (s. f.). *Create an Amazon EBS-backed Linux AMI*.

<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/creating-an-ami-ebs.html>

AWS. (s. f.). *How EC2 Image Builder works.*

<https://docs.aws.amazon.com/imagebuilder/latest/userguide/how-image-builder-works.html>

Krause, J. (2019). *Mastering Windows Server 2019: The Complete Guide for IT Professionals to Install and Manage Windows Server 2019 and Deploy New Capabilities* (2.ª ed.). Packt Publishing.

Lucifredi, F. y Ryan, M. (2018). *AWS System Administration*. O'Reilly Media.

Microsoft. (2018, marzo 30). *Virtual machine extensions and features for Windows*.

<https://docs.microsoft.com/en-us/azure/virtual-machines/extensions/features-windows>

Microsoft. (2018, septiembre 27). *Create a managed image of a generalized VM in Azure*. <https://docs.microsoft.com/en-us/azure/virtual-machines/windows/capture-image-resource>

VMware. (2014, mayo 30). *Choosing QuickPrep or Sysprep to Customize Linked-Clone Machines*. <https://docs.vmware.com/en/VMware-Horizon-7/7.3/horizon-virtual-desktops/GUID-A9C86030-D3E3-4266-9581-E3EAFB8D4BD2.html>