



Institut Puig Castellar
Santa Coloma de Gramenet



Arranque en red con iPXE y iSCSI

CFGS Administració de Sistemes Informàtics i Xarxes

Diego Montesinos Gimeno
Gonzalo Collado Sanahuja

2015/2016 ASIX

3 de Junio de 2016



Aquesta obra està subjecta a una llicència de [Reconeixement-NoComercial-SenseObraDerivada 3.0 Espanya de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

Resumen del proyecto:

En un entorno de trabajo multiusuario, como pueda ser un aula de un centro educativo donde se imparten diversas materias (cada una con sus requisitos de software), sería deseable disponer de un entorno que permitiese centralizar los distintos sistemas operativos a emplear.

El objetivo principal es que un profesor o responsable de impartir el temario, tenga la posibilidad de clonar un s.o. master, y crear una imagen adecuada a una actividad específica, por ejemplo añadiendo el software relativo a la actividad, después pueda desplegar y permitir el acceso a la imagen a los alumnos para cada materia y clase, de esta manera cada alumno podría trabajar en diferentes máquinas manteniendo su configuración y sin perder tiempo en la instalación de software.

Para lograr dicho objetivo es necesario, analizar la tecnología iPXE para conseguir el arranque desde red, profundizar en la configuración de los Target iSCSI en Linux, implementar un servidor SAN iSCSI con todos los servicios necesarios para crear unas adecuadas imágenes master de los sistemas operativos, permitiendo su clonación, preparar el despliegue y repliegue de estas imágenes, e implementarlo en un sistema de ficheros con posibilidad de ahorrar espacio de disco.

Posteriormente se implementará: el desarrollo de scripts para la gestión de imágenes de disco y scripts que gestionen la autenticación de usuarios, permitiendo el arranque de la imagen seleccionada por el usuario. Una vez implementado todo lo anteriormente citado, se realizarían pruebas de rendimiento para comprobar su posible implantación en un entorno de trabajo real.

Abstract:

In a multiuser environment, like a classroom, every subject has his own software requirements and would be desirable to provide of a environment capable of centralize all the operative systems and his related software.

The main goal is that a teacher could have the possibility of cloning a master operative system and create an image fitted for a specific activity, add all the needed software to that image and then export this clone to the students; by doing this, all the students will have a fully working environment ready to use, created only once.

To achieve this goal, we need to: analyze iPXE thecnology to be capable of make network boot, achieve a full understanding of the configuration of Target SCSI on Linux, implement a iSCSI server with all the needed services to make deploys of iSCSI Targets, create master iSCSI Targets to clone and make use of a file system with the capability of save storage space.

Later, we will implement: user authentication, develop of scripts to manage the disk images and its booting. Finally, it will be desirable to make performance tests to check if this serrvice could be used in a real working environment.

Índice

1. Introducción.....	1
1.1. Contexto y justificación del Trabajo.....	1
1.2. Objetivos del Trabajo.....	1
1.3. Enfoque y método seguido.....	1
1.4. Planificación del proyecto.....	1
1.5. Breve resumen de productos obtenidos.....	1
1.6. Breve descripción de los capítulos.....	4
2. Conceptos.....	5
2.1. Preboot Execution Environment.....	5
2.2. SCSI.....	6
2.3. iSCSI.....	8
2.4. SAN (Storage Area Network).....	9
2.5. LIO (Target SCSI).....	10
2.6. Virtualización.....	11
2.7. BTRFS.....	11
3. Requisitos.....	12
4. Análisis de la arquitectura.....	13
4.1. Implementar iPXE mediante chainloading.....	13
4.2. Preparar servidor SAN.....	13
4.3. Scripts para manipular targets.....	13
4.4. Scripts para generar menus iPXE y validación de usuarios.....	13
5. Implementación.....	14
5.1. Configuración del servidor iSCSI.....	14
5.2. Instalación de Sistemas Operativos.....	22
5.3. Despliegue de imágenes clonadas.....	28
5.4. Permisos de acceso a los Targets.....	30
6. Creación de scripts.....	32
6.1. Conceptos targetmgr.....	32
6.2. Directorio <i>/etc/targetmgr</i>	32
6.3. Scripts de targetmgr.....	33
7. Menu iPXE.....	40
7.1. Comandos iPXE generados.....	41
8. Conclusiones.....	43
9. Glosario.....	44
10. Bibliografía.....	47
11. Anexos.....	48
Anexo A. Comandos para el sistema de ficheros ZFS.....	48
Anexo B. Scripts Python.....	52
Anexo C. Scripts PHP.....	71

Lista de figuras

Figura 1,1. Plan de trabajo Parte 1.....	2
Figura 1.2. Plan de trabajo Parte 2.....	3
Figura 2.1. Imagen BIOS.....	5
Figura 2.2. Boot manager.....	6
Figura 2.3. Diagrama de la arquitectura SCSI.....	7
Figura 2.4. Diagrama de la arquitectura iSCSI.....	8
Figura 2.5. SCSI ID y LUN.....	9
Figura 2.6. Tecnologías DAS, NAS y SAN.....	10
Figura 2.7. Arquitectura iSCSI Target.....	10
Figura 5.1. Rango de subred para el servidor iSCSI.....	14
Figura 5.2. Romper bucle iPXE.....	14
Figura 5.3. Configuración sservicio DHCP.....	15
Figura 5.4. Ejemplo de un script de menú iPXE.....	15
Figura 5.5. Ejemplo de un menú iPXE.....	16
Figura 5.6. Ejemplo de configuración del servicio DHCP.....	16
Figura 5.7. Arranque iPXE.....	17
Figura 5.8. Ejecución de comandos iPXE.....	17
Figura 5.9. Inicio de un iSCSI Target.....	17
Figura 5.10. Lista de coandos básicos iPXE.....	18
Figura 5.11. Targetcli: creación del fichero.....	19
Figura 5.12. Targetcli: crear IQN.....	19
Figura 5.13. Targetcli: crear portal.....	20
Figura 5.14. Targetcli: crear LUN.....	20
Figura 5.15. Targetcli: demo mode.....	20
Figura 5.16. Targetcli: guardar configuración.....	20
Figura 5.16. Targetcli: ficheros de backup.....	21
Figura 5.17. Targetcli: fichero de configuración.....	21
Figura 5.18. Targetcli: targets.....	21
Figura 5.19. Targetcli: preparación para instalar un sistema operativo.....	22
Figura 5.20. Instalación Fedora: selección de disco duro.....	22
Figura 5.21. Instalación SO: añadir disco iSCSI.....	23
Figura 5.22. Instalación Fedora: descubrimiento Target iSCSI.....	23
Figura 5.23. Instalación Fedora: selección de Target iSCSI.....	24
Figura 5.24. Instalación Fedora: Destino de la instalación.....	25
Figura 5.25. Instalación Fedora: Destino de la instalación.....	25
Figura 5.26. Instalación Ubuntu: Selección de disco.....	26
Figura 5.27. Instalación Ubuntu: nombre del iSCSI Initiator.....	26
Figura 5.28. Instalación Ubuntu: establecer dirección IP del servidor iSCSI.....	27
Figura 5.29. Instalación Ubuntu: login en el iSCSI Target.....	27
Figura 5.30. Instalación Ubuntu: selección de iSCSI Target.....	28
Figura 5.31. Archivo iscsi.initramfs en Ubuntu 16.04.....	28
Figura 5.32. Archivo iscsi.initramfs en Ubuntu 16.04.....	29
Figura 5.33. Actualización de initramfs.....	29
Figura 5.34. Copia manual de un fichero de imagen.....	29
Figura 5.35. Modificación del fichero ibft.conf en Fedora.....	29
Figura 5.36. Actualización de GRUB en Fedora 21.....	30
Figura 5.37. Targetcli: configuración de permisos de acceso para un Target.....	30
Figura 5.38. Targetcli: fichero de configuración.....	31
Figura 5.39. Ejemplo de script de menú iPXE con login.....	31
Figura 5.40. iPXE: pantalla de login.....	31
Figura 6.1. Fichero de configuración.....	33
Figura 6.2. Fichero users.....	33
Figura 6.3. Script tgtcreatemaster: parámetros.....	34
Figura 6.4. Script tgtclonemaster: parámetros.....	35
Figura 6.5. Script tgtcloneimage: parámetros.....	35
Figura 6.6. Targetcli: estructura de los Targets.....	37
Figura 6.7. Estructura de directorios.....	37
Figura 6.8. Ayuda script tgtdeletemasater.....	38
Figura 6.9. Ayuda script tgtdeleteimage.....	39
Figura 7.1. Pantalla autenticación iPXE.....	40
Figura 7.2. Selección de imagenes publicadas para master.....	40
Figura 7.3. Selección de imagenes publicadas para alumno3.....	40

1. Introducción

1.1. Contexto y justificación del Trabajo

Este proyecto nos permitirá desarrollar nuestras capacidades en el ámbito de la administración de sistemas operativos y servicios de red. Nos motiva la creación de un servidor de sistemas operativos en red que permita trabajar con diferentes imágenes de disco en una misma máquina cliente y realizar las pruebas necesarias en vista a una implantación en producción de este sistema.

1.2. Objetivos del Trabajo

- Implementar un servidor SAN iSCSI.
- Hacer uso de la tecnología iPXE y de las imágenes almacenadas en el servidor iSCSI para realizar arranques en red.
- Implementar la autenticación de usuarios, de manera que a un usuario determinado se le sirva una imagen específica.
- Desarrollo de herramientas sencillas para la gestión de las imágenes y los usuarios (mediante scripts).

1.3. Enfoque y método seguido

Partiremos de un servidor con el sistema de ficheros BTRFS ya existente en el centro, en el que prepararemos diversos sistemas operativos, para arrancar a través de iSCSI. A partir de estas instalaciones maestras, realizaremos clonaciones para cada usuario e implementaremos el servicio iPXE, haciendo posible el arranque desde red. El siguiente paso será configurar la autenticación de usuarios para posteriormente implementar menús y que los usuarios puedan elegir su imagen de arranque correspondiente, después de haberse identificado como usuarios válidos.

1.4. Planificación del proyecto

Para la elaboración del plan de trabajo del proyecto hemos hecho uso de la herramienta Planner para la creación del diagrama de Gantt, donde reflejamos las diferentes tareas y los tiempos previstos para su finalización.

1.5. Breve resumen de productos obtenidos

El producto final será un servidor SAN iSCSI plenamente funcional, capaz de servir los targets adecuados a los clientes de una LAN. Además de las herramientas para gestionar despliegues y repliegues de targets, y que implementan la clonación en el sistema de ficheros, junto a un sistema de validación de usuarios y los menús correspondientes en la selección de imágenes de arranque.

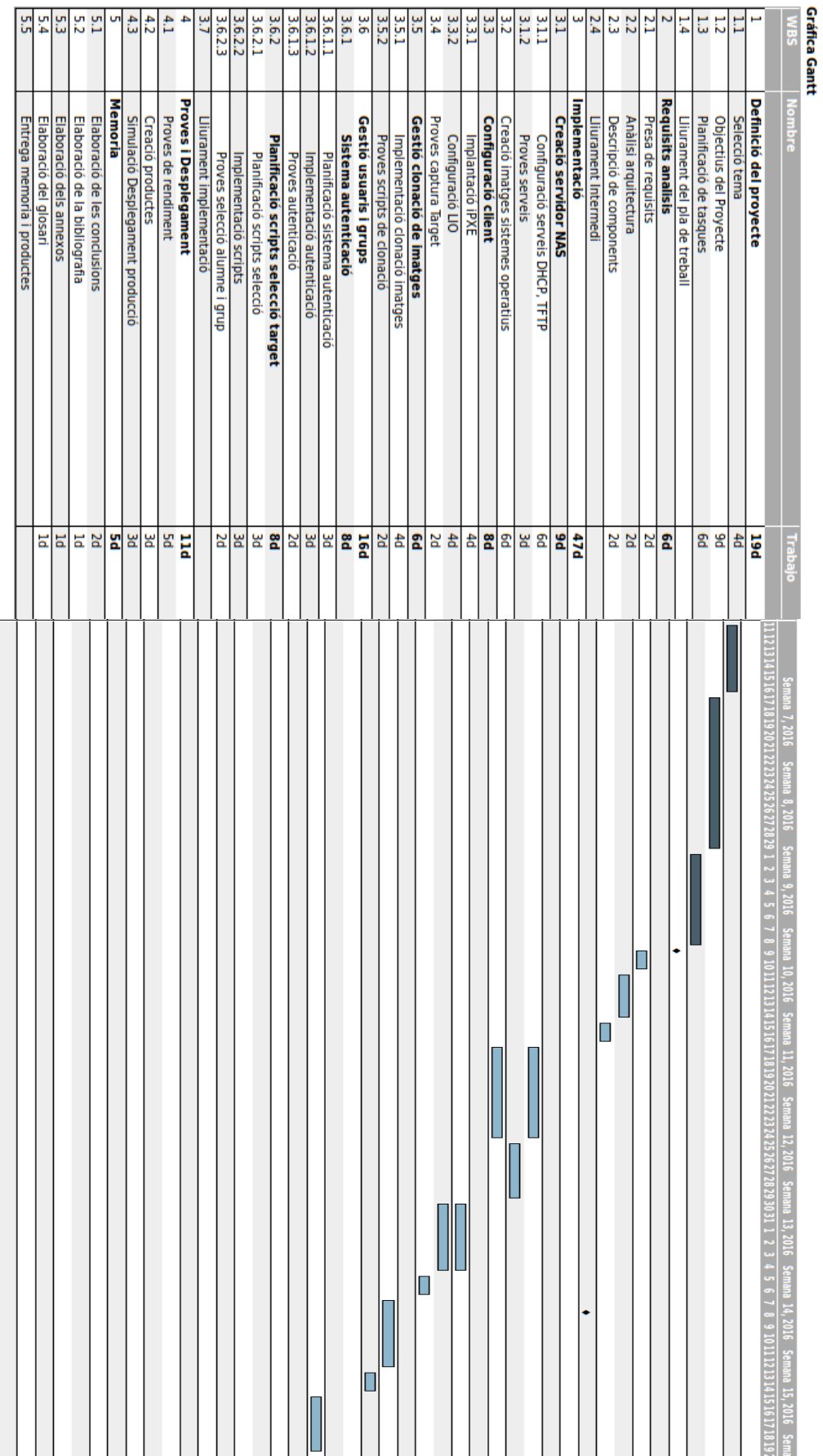


Figura 1,1. Plan de trabajo Parte 1

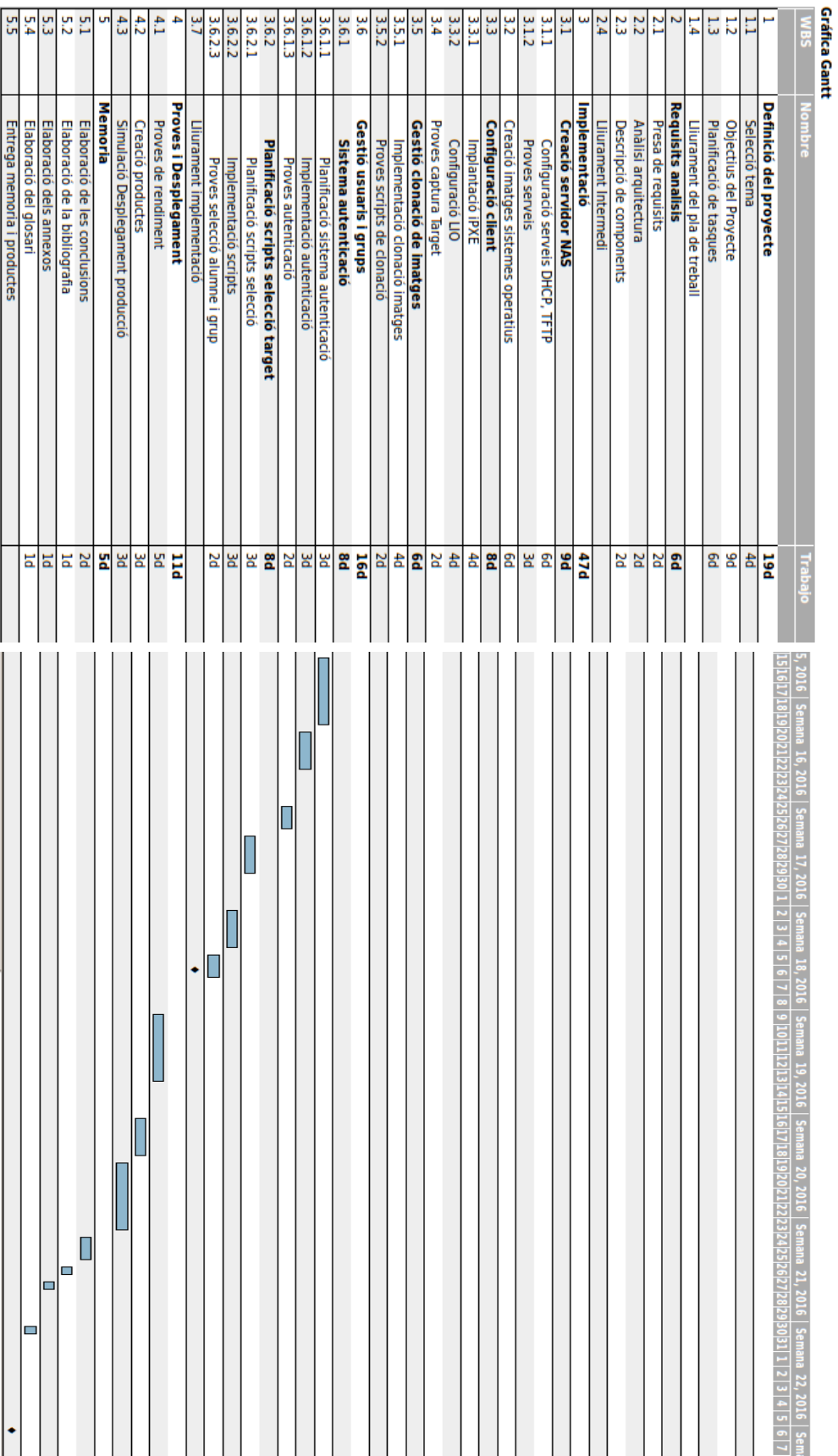


Figura 1.2. Plan de trabajo Parte 2

1.6. Breve descripción de los capítulos

Mostraremos ahora una breve descripción de los capítulos restantes:

- **Conceptos** : Definición de conceptos básicos para desarrollar el proyecto, tales como PXE, iPXE, SCSI, iSCSI, SAN LIO, y BTRFS
- **Requisitos**: Requisitos necesarios para implementar el servidor SAN iSCSI.
- **Análisis de la arquitectura**: Análisis del software y hardware necesarios para el desarrollo del proyecto.
- **Implementación**: Pasos a seguir para la configuración e implementación del servicio. Tanto en la preparación del servidor, como en la instalación de los sistemas operativos.
- **Creación de scripts**: Scripts para gestionar el servicio, y automatizar tareas más comunes.
- **Menú iPXE**: Código php para generar comandos iPXE.
- **Conclusiones**: Conclusiones finales y posibles mejoras a realizar.
- **Glosario**: Diccionario de términos relacionados.
- **Anexos**: Código fuente de los scripts.

2. Conceptos

2.1. Preboot Execution Environment

PXE es uno de los componentes de la especificación WfM de Intel. Permite iniciar una estación de trabajo desde un servidor en una red antes de arrancar el sistema operativo en el disco duro local. Una estación de trabajo habilitada con PXE, conecta su tarjeta a la LAN a través de una conexión a la red. Debido a que un administrador de red no tiene que desplazarse físicamente a la estación de trabajo, puede manualmente desde un servidor a través de red, arrancarla, iniciarla y cargar en el PC un sistema operativo u otro tipo de software, como programas de diagnóstico.

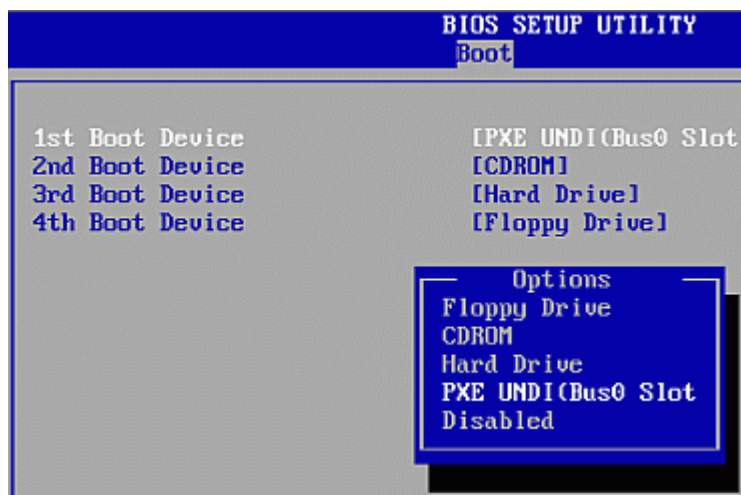


Figura 2.1. Imagen BIOS

NOTA: PXE es un elemento obligatorio de la especificación Wfm, debe ser soportado por la BIOS de la estación de trabajo y su tarjeta de red NIC.

2.1.2. iPXE

Es una implementación open source del firmware cliente y bootloader de PXE, creada en 2010 como fork de un proyecto anterior gPXE. Es posible utilizarlo para permitir que equipos sin soporte PXE arranquen desde la red o para extender la implementación de clientes PXE soportando clientes adicionales.

Mientras que los clientes estandarizados PXE usan TFTP para transferir los datos, los clientes con el firmware iPXE añaden la posibilidad de recuperar datos a través de otros protocolos, incluyendo HTTP, iSCSI, ATA over Ethernet (AoE) y Fibre Channel over Ethernet (FCoE). También si el hardware lo soporta puede utilizarse un enlace Wi-Fi.

Implementación PXE en iPXE

iPXE puede ser ejecutado en un ordenador de dos maneras distintas:

- Sustituyendo la ROM estándar PXE existente (re-flashing) de la tarjeta de red (NIC)
- Arrancando con PXE y mediante *chainloading*, cargar el binario de iPXE sin la necesidad de sustituir la ROM de la NIC (re-flashing).

La imagen del firmware iPXE lleva embebido su script de configuración. Por lo tanto cualquier cambio en la configuración requerirá una nueva actualización de la ROM.

iPXE implementa su propia pila PXE, utilizando el controlador de la tarjeta de red proporcionado por iPXE, o el controlador PXE UNDI (*Universal Network Driver Interface*) cargado desde una ROM estándar PXE. La implementación de una pila PXE independiente permite a los clientes sin la ROM PXE estándar en sus tarjetas de red usar una pila iPXE alternativa.

Boot manager

Aunque su función básica es implementar una pila PXE, iPXE puede también ser utilizado como gestor de arranque de red con capacidades limitadas de interacción con los usuarios finales basadas en menús. Puede recuperar archivos de arranque usando múltiples protocolos de red, como TFTP, NFS, HTTP o FTP.

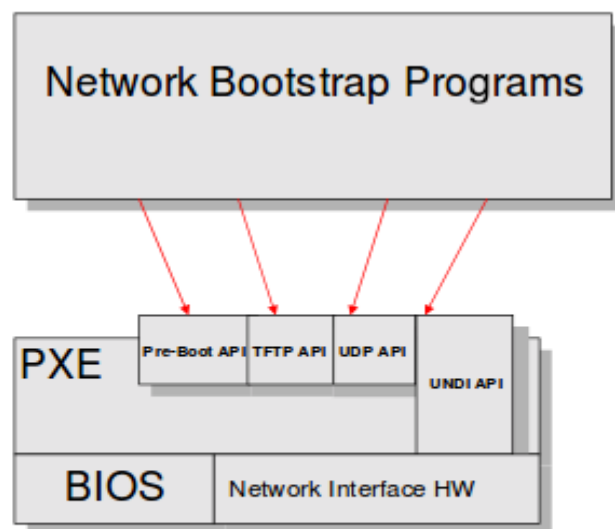


Figura 2.2. Boot manager

2.2. SCSI

Small Computer System Interface (Interfaz de sistema para computadoras pequeñas) es un conjunto de estándares empleados para la conexión física y la transmisión de datos entre computadoras y periféricos. Aunque SCSI permite la conexión de diversos tipos de dispositivos periféricos (como por ejemplo, unidades de CD/DVD, impresoras o scanners, su uso más extendido es el de conectar dispositivos de almacenamiento).

El estándar SCSI define una serie de comandos de control para el establecimiento de conexión entre un dispositivo cliente y un dispositivo servidor, conocidos en la terminología de SCSI como SCSI Initiator (cliente) y SCSI Target (servidor).

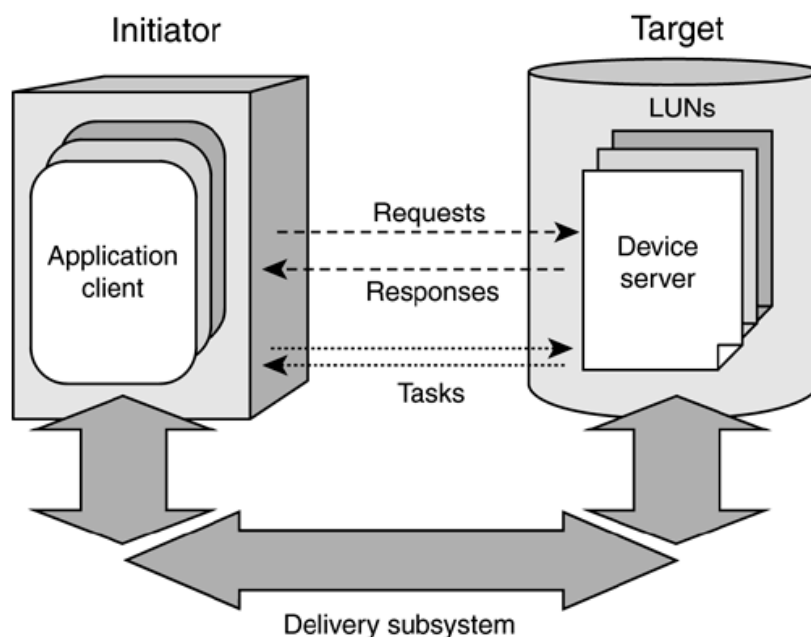


Figura 2.3. Diagrama de la arquitectura SCSI

Cada dispositivo SCSI está identificado por un *Logical Unit Number* (LUN), un número que identifica de manera inequívoca un dispositivo de almacenamiento, sea éste un dispositivo físico, un volumen lógico en un array o una partición de un disco duro.

2.2.1. Interfaces y tecnologías

Parallel SCSI (SPI): la interfaz original del protocolo SCSI, capaz de transferir datos a velocidades comprendidas entre 5 y 320 MB/s

Serial Attached SCSI (SAS): reemplazo de la interfaz SPI; protocolo para la conexión de dispositivos de almacenamiento y el transporte de datos, permite velocidades de transmisión comprendidas entre los 3 Gbits/s(SAS-1) y los 12 Gbits/s (SAS-3)

USB Attached SCSI (UAS): protocolo para la transferencia de datos de dispositivos de almacenamiento USB. Introducido como parte de la versión 3 del protocolo USB, hace uso del conjunto de comandos SCSI.

Fibre Channel (FC): tecnología de red que proporciona altas velocidades de transferencia (entre 2 y 16 Gbits/s) y que se ha convertido en el tipo de conexión más común en las redes de almacenamiento (Storage Area Network, SAN). FC utiliza el protocolo de transporte Fibre Channel Protocol (FCP) para transportar por red los comandos SCSI.

2.2.2. Comandos SCSI

Aunque originalmente diseñada para el bus paralelo, la arquitectura de comandos SCSI ha sido portada a otros sistemas de transmisión de datos, como Fibre Channel, o iSCSI.

Al estar basado en un modelo cliente-servidor, el protocolo SCSI necesita un SCSI Initiator que dé inicio a la conexión con un SCSI Target. Los comandos del protocolo SCSI son transferidos en un bloque descriptor de comandos (command descriptor block, CDB), consistente en un Byte con el código de operación seguido por 5 o más Bytes que contienen parámetros específicos del comando.

Existen cuatro categorías diferentes de comandos de control SCSI :

- N (non-data)
- W (escritura de datos desde el SCSI Initiator al SCSI Target)
- R (lectura de datos)
- B (bidireccional)

NOTA: Al no ser el objetivo principal de esta memoria, no exponemos la lista completa de comandos, puede ser consultada en la siguiente url: https://en.wikipedia.org/wiki/SCSI_command

2.3. iSCSI

Internet SCSI es una implementación sobre redes TCP/IP del protocolo SCSI, permitiendo la conexión de hosts con dispositivos de almacenamiento a través de la red.

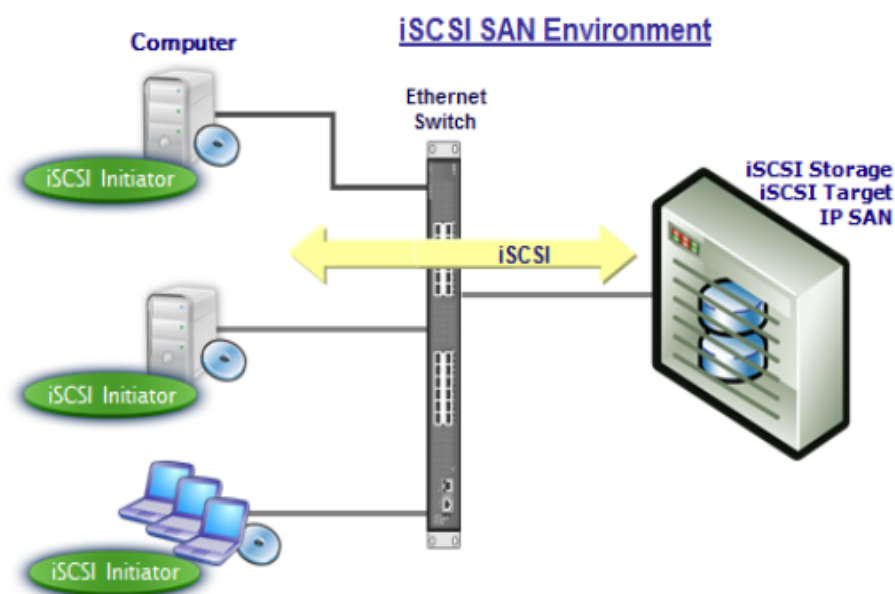


Figura 2.4. Diagrama de la arquitectura iSCSI

El protocolo iSCSI permite la conexión de diversos iSCSI Initiators a uno o varios iSCSI Targets, generalmente agrupados en una *Storage Area Network (SAN)*; los iSCSI Targets serán reconocidos por los iSCSI Initiators como si fueran dispositivos físicos conectados a ellos directamente.

iSCSI no sólo permite la exportación de unidades de almacenamiento completas, sino que también exporta particiones de un disco, volúmenes dentro de un RAID/LVM o incluso ficheros de bloque, y todos estos dispositivos son tratados como discos duros independientes.

2.3.1. Targets e Initiators

En el Apartado 2.2 hemos ofrecido una breve descripción de los Targets y los Initiators, elementos imprescindibles en una conexión SCSI, pero ahora profundizaremos más en la descripción de estos términos.

En el protocolo iSCSI, el Initiator da inicio a la sesión, transmitiendo comandos SCSI a través de la red y accediendo a los dispositivos de almacenamiento presentes en un servidor SAN. Un SAN puede estar compuesto por uno o más discos físicos y cada uno de estos dispositivos físicos puede a su vez estar compuesto de varias particiones. A cada dispositivo que soporta operaciones de lectura/escritura (discos duros físicos, particiones de un disco duro, ficheros de bloques, particiones LVM/RAID) se le asigna un identificador conocido como LUN (*Logical Unit Number*). Un LUN es un número que identifica de manera única e inequívoca a una unidad lógica (dispositivo de almacenamiento)

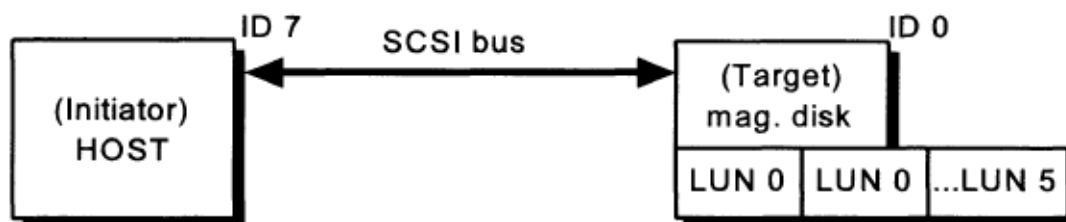


Figura 2.5. SCSI ID y LUN

El protocolo SCSI establece también un identificador único para cada dispositivo conectado; este dispositivo puede contener múltiples unidades lógicas cada una de ellas identificadas con su LUN particular, en nuestro caso solamente incluimos solamente un LUN para cada target iSCSI. El Initiator usará el LUN correspondiente para identificar el volumen lógico al que debe dirigir los comandos.

2.4. SAN (Storage Area Network)

SAN: La tecnología SAN proporciona acceso a los dispositivos a nivel de bloque, es decir: los clientes de una red SAN acceden a los dispositivos disponibles como si estuvieran directamente conectados, viéndolos como dispositivos físicos locales, realizando cualquier acción sobre ellos, como particionarlos o configurar RAID. Esta tecnología, además del protocolo iSCSI que utilizaremos, hace uso de ATA over Ethernet(AoE), Fibre Channel Protocol (FCP).

DAS: Es el método tradicional de conexión punto a punto de dispositivos de almacenamiento a un PC, en el que el disco duro, o conjunto de ellos, están conectados física y directamente a la máquina que accede a ellos, sin ningún tipo de dispositivo intermedio. Es un acrónimo relativamente nuevo, creado en oposición a NAS y SAN.

NAS: Almacenamiento conectado en red; servidor de almacenamiento de datos a nivel de ficheros, ofrece servicios de compartición de ficheros entre los hosts de una red. Hace uso de los protocolos SMB, Network File System (NFS) o SFTP. Para los clientes, los dispositivos de almacenamiento NAS aparecen como sistemas de ficheros compartidos.

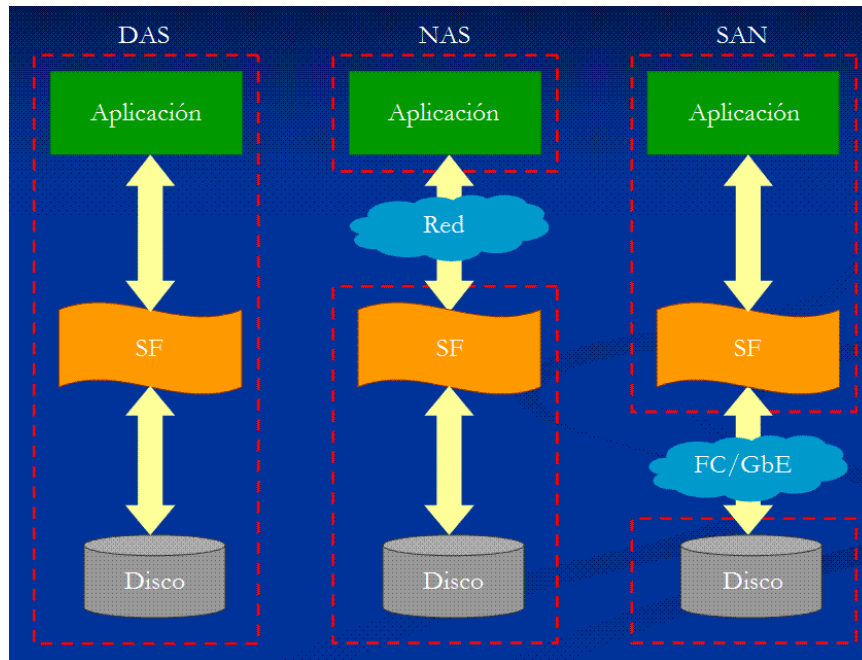


Figura 2.6. Tecnologías DAS, NAS y SAN

2.5. LIO (Target SCSI)

Es la implementación open-source de SCSI target, que se ha convertido en el estándar incluido en el kernel de Linux. Internamente, LIO no inicializa sesiones, en su lugar proporciona uno o más LUNs (Logical Unit Number), esperando los comandos SCSI desde un inicializador SCSI y desarrollando las transferencias de entrada y salida de datos requeridas.

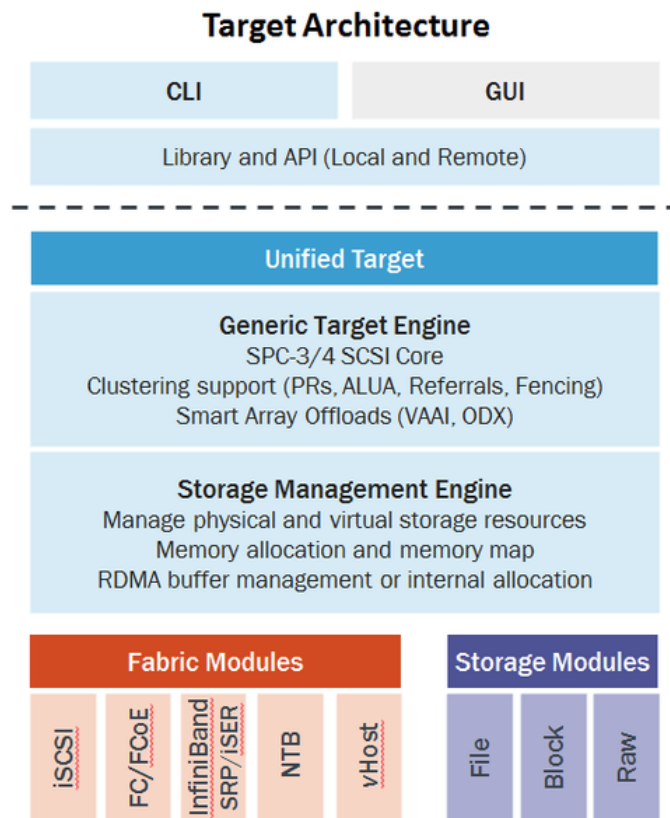


Figura 2.7. Arquitectura iSCSI Target

2.6. Virtualización

Kernel-based Virtual Machine(KVM) es una infraestructura de virtualización para el núcleo de Linux, que lo que transforma en un *hypervisor*. Está presente desde la versión 2.6.20 del kernel. KVM requiere una CPU con extensión hardware de virtualización.

Hypervisor: o Monitor de Máquina Virtual (Virtual Machine Monitor, VMM), software, hardware o firmware que permite la creación y ejecución de máquinas virtuales.

2.7. BTRFS

El sistema de archivos BTRFS “*B-tree file system*”, es un sistema de archivos, habitualmente considerado experimental, aunque en estos momentos según indican sus desarrolladores: “el formato de disco del sistema de archivos ya no es inestable, y no se espera que cambie si no existen fuertes razones para hacerlo. La base de código Btrfs está en fuerte desarrollo. Se está haciendo todo lo posible para mantenerlo estable y rápido. Debido a la velocidad de desarrollo rápido, el desarrollo del sistema de archivos mejora notablemente con cada nueva versión de Linux, por lo que es recomendable hacer funcionar el núcleo más moderno posible.”

Inicialmente desarrollado por Oracle, está en desarrollo continuo. Utiliza el concepto de *extends* como el sistema de archivo *ext4* y el de *subvolúmenes*. Este método también permite la creación de instantáneas (*snapshots*) de un árbol, la imagen exacta del árbol en un instante determinado, permitiendo comprobar las modificaciones o volver al estado inicial, como con *LVM*.

Establece un sistema de datos y de metadatos, especialmente con checksums que permiten averiguar si un archivo se ha corrompido. Otra ventaja es que permite redimensionar el sistema de archivos en caliente, incluso reducirlo. Se puede organizar en varios volúmenes, pareciéndose a soluciones del tipo *LVM*.

3. Requisitos

En el centro educativo se dispone de aulas informáticas en las que trabajan diferentes grupos y se imparten diferentes materias. Pese a que las máquinas virtuales son una excelente herramienta de trabajo, sería conveniente disponer de un sistema operativo individual por alumno y materia. Habiendo tantas materias no resulta razonable instalar localmente los sistemas operativos.

Objetivos:

- Obtener un servidor SAN iSCSI.
- Obtener un sistema que permita arrancar mediante PXE los ordenadores utilizando las imágenes del servidor SAN.
- Permitir que los usuarios escojan en el momento del arranque: grupo, usuario e imagen.
- Permitir que los usuarios se identifiquen con usuario y contraseña en el momento del arranque para evitar accesos indebidos.
- Permitir que a partir de una imagen maestra, se puedan realizar clones para varios usuarios.
- Desarrollar unos shell scripts, o una pequeña herramienta de administración web que permita la gestión de las imágenes.

4. Análisis de la arquitectura

4.1. Implementar iPXE mediante chainloading

Como disponemos de un gran número de ordenadores con la implementación PXE (tarjetas de red con la ROM PXE de Intel), evitaremos flashear cada NIC una por una y cargaremos iPXE usando chainloading.

Necesitaremos una ubicación donde colocar una copia de iPXE en nuestro servidor TFTP. Los ordenadores deberán descargar esta copia iPXE desde el servidor TFTP cada vez que arranquen.

4.2. Preparar servidor SAN

Virtualizar el servidor SAN, y preparar los volúmenes y carpetas con el sistema de ficheros Btrfs. Crear los iSCSI LUN y Target en el servidor, realizar el proceso de configuración de montaje y puesta en marcha de iSCSI Target en Linux. También tendremos que activar el servicio iSCSI en el servidor.

4.3. Scripts para manipular targets

Conjunto de herramientas para manipular targets iSCSI, realizadas en python utilizando el modulo *rstlib*. Estos scripts además de publicar los targets, realizarán las operaciones de clonación y eliminación sobre los ficheros de las imagenes de cada target.

4.4. Scripts para generar menus iPXE y validación de usuarios.

Sistema de validación de los usuarios en el sistema, junto a los scripts PHP que generarán los comandos iPXE, y que mostrarán solamente las imagenes disponibles para cada usuario.

5. Implementación

5.1. Configuración del servidor iSCSI

5.1.1. Protocolo DHCP

Una vez instalado el software anteriormente mencionado, necesitamos instalar el firmware iPXE que será enviado mediante FTP al cliente cuando éste solicite un inicio en red; el software puede ser descargado desde la dirección <http://boot.ipxe.org/undionly.kpxe>. El fichero lo almacenamos en el directorio `/var/lib/tftpboot` del servicio FTP. El fichero `undionly.kpxe` será el que permita al cliente enviar y recibir comandos ipxe para acceder a los dispositivos iSCSI.

Configuramos el servidor DHCP para que las máquinas clientes sean capaces de conectar al servidor y acceder a los dispositivos iSCSI que exportaremos con el servicio Targetcli. Para ello configuramos un rango de direcciones IP para los clientes dentro de la red 10.0.0.0/8:

```
subnet 10.0.0.0 netmask 255.0.0.0 {
    range 10.1.0.100 10.1.0.150;
    option routers 10.1.0.1;
```

Figura 5.1. Rango de subred para el servidor iSCSI

Para servir un fichero a través de FTP, bastaría con incluir en la configuración DHCP la línea `filename "undionly.kpxe"`, pero si lo hacemos de esta manera, cuando arrancásemos desde un cliente, el arranque en red entraría en un bucle del que no podría salir y en el que constantemente estaría solicitando el firmware iPXE al servidor, para después solicitar IP al servidor DHCP y volver a descargar el fichero `undionly.kpxe` indefinidamente.

Este bucle se rompe con la inclusión de las siguientes líneas:

```
if exists user-class and option user-class = "iPXE"
{
    filename "menu";
}
else
{
    filename "undionly.kpxe";
}
```

Figura 5.2. Romper bucle iPXE

El condicional también nos permite incluir un sencillo menú de arranque iPXE que se ejecutará en el momento en el que el cliente haya descargado el software `undionly.kpxe` (del desarrollo del script de menú nos ocuparemos más adelante).

El fichero completo de configuración de DHCP (`/etc/dhcp/dhcpd.conf`) queda de la siguiente manera:

```

subnet 10.0.0.0 netmask 255.0.0.0{
    range 10.1.0.100 10.1.0.150;
    option routers 10.1.0.1;
    next-server 10.1.0.1;

    if exists user-class and option user-class = "iPXE"
    {
        filename "menu";
    }
    else
    {
        filename "undionly.kpxe";
    }
}

```

Figura 5.3. Configuración servicio DHCP

Aunque aún estamos desarrollando los scripts necesarios para la definitiva implementación del servicio, haremos una breve mención a los scripts iPXE. Para ilustrar dicho tema, creamos un sencillo script que nos mostrará un menú con dos opciones, correspondientes a los sistemas operativos que tenemos preparados para su exportación. Por el momento, disponemos de un fichero donde hemos instalado un Fedora 21 y que hemos clonado mediante la capacidad de BTRFS de realizar instantáneas de un fichero, mediante el comando `cp -reflink="always"`; hemos creado en Targetcli un Target para cada fichero (Ver configuración Targetcli)

El código del script que nos permitirá conectar a los discos duros es el siguiente:

```

#!ipxe

dhcp

#Menú

menu Disk Boot Menu
item fedoramaster Fedora 21 (master)
item fedoraclon Fedora 21 (clon)

choose option && goto ${option}

:fedoramaster
sanboot iscsi:10.1.0.1:::iqn.2016-04.org.example:fedora
:fedoraclon
sanboot iscsi:10.1.0.1:::iqn.2016-04.org.example:fedora21

```

Figura 5.4. Ejemplo de un script de menú iPXE

Etiqueta *menu* - indica el inicio de un menú PXE

Etiqueta *item* - inserción de un elemento seleccionable

`choose option && goto ${option}` - la elección de un ítem se recoge en la variable *option*, para después ejecutar el comando asociado a ese ítem.

Cuando el cliente conecte con el servidor se le mostrará la siguiente pantalla:



Figura 5.5. Ejemplo de un menú iPXE

Para realizar pruebas rápidas de conectividad entre el cliente y el servidor, podemos obviar toda la parte del código condicional, y únicamente incluir la línea `filename "undionly.kpxe"` tras la configuración de las direcciones IP, pero teniendo en cuenta que, una vez que el cliente haya establecido conexión con el servidor, deberemos detener el bucle manualmente pulsando `Ctrl+B`, lo que nos dará acceso al "prompt" iPXE, donde ya podremos ejecutar los comandos iPXE pertinentes.

Los Targets iSCSI pueden ser accedidos mediante la introducción de comandos PXE una vez que se ha establecido la conexión con el servidor. Este no es un método práctico ni de fácil empleo para usuarios legos en la materia, pero para realizar pruebas y familiarizarse con el sistema puede ser útil.

Para un acceso manual, el fichero de configuración del servicio DHCP se hace un poco más simple pues podemos prescindir del condicional que rompe el bucle de peticiones DHCP/Undionly.kpxe (detendremos el bucle de manera manual), además de que no necesitaremos un menú personalizado de arranque.

El fichero de configuración de DHCP queda como sigue:

```
subnet 10.0.0.0 netmask 255.0.0.0{
    range 10.1.0.100 10.1.0.150;
    option routers 10.1.0.1;
    next-server 10.1.0.1;
    filename "undionly.kpxe";
}
```

Figura 5.6. Ejemplo de configuración del servicio DHCP

Cuando arranca la máquina cliente, ésta entra en un bucle continuo del que se la puede sacar presionando `Ctrl-B`;

```

iPXE 1.0.0+ (55e4) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

net0: 08:00:27:0c:d4:43 using undionly on UNDI-PCI00:03.0 (open)
  [Link:up, TX:0 TXE:0 RX:0 RXE:0]
Configuring (net0 08:00:27:0c:d4:43)..... ok
net0: 10.1.0.107/255.0.0.0 gw 10.1.0.1
Next server: 10.1.0.1
Filename: undionly.kpxe
tftp://10.1.0.1/undionly.kpxe... ok
undionly.kpxe : 64292 bytes [PXE-NBP]
PXE->EB: !PXE at 9138:07D0, entry point at 9138:0187
         UNDI code segment 9138:0872, data segment 91C0:3000 (580-595kB)
         UNDI device is PCI 00:03.0, type DIX+802.3
         580kB free base memory after PXE unload
iPXE initialising devices...ok

iPXE 1.0.0+ (55e4) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

Press Ctrl-B for the iPXE command line...

```

Figura 5.7. Arranque iPXE

De este modo, entramos en la línea de comandos iPXE para proceder a la conexión y arranque de un Target:

```

iPXE 1.0.0+ (55e4) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

iPXE> dhcp
Configuring (net0 08:00:27:0c:d4:43)..... ok
iPXE> route
net0: 10.1.0.112/255.0.0.0 gw 10.1.0.1
iPXE> sanboot iscsi:10.1.0.1:::iqn.2016-04.org.example:fedora_

```

Figura 5.8. Ejecución de comandos iPXE

Primero, solicitamos una dirección IP al servidor iSCSI para poder acceder al Target (podemos comprobar la ruta con el comando Route) para después proceder con el comando de carga:

sanboot iscsi:IPservidor:<puertoTCP>:<>:<LUN>:iqn.XXXX-XX.dominio.nombre:nombre del target

El comando anterior devuelve la siguiente salida y comienza el proceso de arranque del SO que tengamos instalado en el Target:

```

iPXE> sanboot iscsi:10.1.0.1:::iqn.2016-04.org.example:fedora
Registered SAN device 0x80
Booting from SAN device 0x80

```

Figura 5.9. Inicio de un iSCSI Target

iPXE ofrece una serie de comandos básicos que pueden ser consultados con el comando *help*:

```
iPXE 1.0.0+ (55e4) -- Open Source Network Boot Firmware -- http://ipxe.org
Features: DNS HTTP iSCSI TFTP AoE ELF MBOOT PXE bzImage Menu PXEXT

iPXE> help

Available commands:

config      cpuid      dhcp       pxeps      ifopen
ifclose     ifstat    ifconf     imgfetch   module
initrd      kernel    chain      imgselect  imgload
imgargs     imgexec   boot       imgstat    imgfree
login       menu      item       choose     show
set         clear     read       inc        reboot
route       sanhook   sanboot    sanunhook  goto
prompt     sync      autoboot   echo       exit
isset     iseq     sleep      help       shell
```

Figura 5.10. Lista de comandos básicos iPXE

5.1.2 Protocolo TFTP:

El servicio TFTP no requiere una configuración compleja, pues sólo lo emplearemos para servir inicialmente el software de arranque en red iPXE.

Teniendo esto en cuenta, tan solo necesitamos copiar el fichero `undionly.kpxe` (o descargar mediante `wget http://boot.ipxe.org/undionly.kpxe`) al directorio `/var/lib/tftpboot` e indicar en el fichero `/etc/dhcp/dhcpd.conf` que el servidor TFTP tiene por IP la dirección 10.1.0.1 (que en nuestro caso es el mismo servidor iSCSI) con la opción `next-server`.

5.1.3 Targetcli

Targetcli es una herramienta que nos permite configurar los Targets que se exportarán.

Primero, necesitamos crear un directorio específico donde almacenar las imágenes que darán soporte a los discos scsi; en nuestro caso, creamos la estructura `/opt/pool/master` para almacenar las imágenes “originales” (en el momento de desplegar los Targets iSCSI, se crearán clones para cada usuario, de manera que siempre exista una copia inalterada de la instalación original)

En la shell de Targetcli, creamos la imagen en `/backstores/fileio`, con el comando

```
create <identificador> </ruta/de/almacenaje> <tamaño>G
```

```
/> cd /backstores/fileio/  
/backstores/fileio> create ubuntu16 /opt/pool/master/ubuntu16.img 15G  
Using buffered mode.  
Created fileio ubuntu16.
```

Figura 5.11. Targetcli: creación del fichero

Cabe señalar que Targetcli crea todos los Targets por defecto con el método non-buffered para la escritura de datos sobre el Fileio. Esto implica que, cuando el Target está en uso en una máquina cliente, todos los datos se escriben sobre el Target directamente, no pasan por un búffer intermedio que posteriormente volcará los datos sobre el disco. Los cambios en disco se producen en tiempo real por lo que se minimiza la posibilidad de pérdida de datos, en cambio, una desconexión imprevista del Target iSCSI y una caché no volcada a disco, puede dañar el sistema de ficheros y provocar la pérdida de datos.

Activar/desactivar el modo *buffered*:

create <target> /ruta/fichero.img nG buffered=[true | false]

El siguiente paso consiste en crear el identificador del Target. Este identificador o IQN es lo que permitirá a los clientes o Initiators reconocer nuestros ficheros como dispositivos iSCSI válidos para la conexión.

El nombre que le otorguemos al Target puede ser de nuestra invención, pero debe seguir una estructura concreta para ser válido:

iqn.<fecha desde la cual es válido el Target>.<nombre de dominio invertido>:<nombre de la imagen>

Targetcli otorgará un número de serie o **WWN** por defecto a nuestro Target, además de crear automáticamente un **Target Portal Group (TPG)** asociado al Target.

```
/> cd iscsi  
/iscsi> create iqn.2016-04.org.example:ubuntu16  
Created target iqn.2016-04.org.example:ubuntu16.  
Selected TPG Tag 1.  
Created TPG 1.
```

Figura 5.12. Targetcli: crear IQN

A continuación, necesitamos especificar una dirección IP y un puerto TCP (un Portal) a través de los cuales los Initiators serán capaces de conectarse al Target. La dirección IP será la del servidor que aloja el servicio, mientras que el puerto será el 3260, el puerto por defecto del servicio.


```

/> cd iscsi/iqn.2016-04.org.example:ubuntu16/
/iscsi/iqn.20...mple:ubuntu16> cd tpg1/portals
/iscsi/iqn.20.../tpg1/portals> create 10.1.0.1
Using default IP port 3260
Created network portal 10.1.0.1:3260.

```

Figura 5.13. Targetcli: crear portal

Cada Target necesita como mínimo un LUN(Logical Unit Number) para identificar al dispositivo. Un Target puede contener diversos dispositivos, cada uno identificado por un LUN, en caso de sólo existir un dispositivo, se le asignará de manera automática un LUN 0. El LUN queda asociado de manera inequívoca a un fichero.

```

/iscsi/iqn.20.../tpg1/portals> cd ..
/iscsi/iqn.20...ubuntu16/tpg1> cd luns
/iscsi/iqn.20...u16/tpg1/luns> create /backstores/fileio/ubuntu16
Selected LUN 0.
Created LUN 0.

```

Figura 5.14: Targetcli: crear LUN

Pueden asignarse permisos de acceso para cada Target de manera individual, pudiendo especificar que es necesario un nombre de usuario y una contraseña para conectar con el dispositivo.

Por el momento, no asignaremos ninguna medida de control de acceso a los Targets pues estamos en fase de pruebas del servicio, por lo que configuraremos el servidor ISCSI en modo de prueba, por lo que cualquier cliente podrá, sin necesidad de realizar login, acceder al Target.

```

/iscsi/iqn.20...ubuntu16/tpg1> set attribute authentication=0 demo_mode_write_protect=0 generate_node_acls=1 cache_dynamic_acls=1
Parameter demo_mode_write_protect is now '0'.
Parameter authentication is now '0'.
Parameter generate_node_acls is now '1'.
Parameter cache_dynamic_acls is now '1'.

```

Figura 5.15. Targetcli: demo mode

El último paso es guardar la configuración del servicio con el comando saveconfig

```

/> saveconfig
Save configuration? [Y/n]:
Performing backup of startup configuration: /var/target/backup-2016-04-24_23:25:19.lio
Saving new startup configuration

```

Figura 5.16. Targetcli: guardar configuración

El servicio Target almacena backups de las configuraciones de los targets declarados en el directorio `/var/target`

```
-rw-r--r-- 1 root root 2492 abr 21 16:29 backup-2016-04-21_16:29:57.lio
-rw-r--r-- 1 root root 2557 abr 21 16:43 backup-2016-04-21_16:43:23.lio
-rw-r--r-- 1 root root 3230 abr 21 16:47 backup-2016-04-21_16:47:17.lio
-rw-r--r-- 1 root root 3238 abr 21 16:52 backup-2016-04-21_16:52:46.lio
```

Figura 5.16. Targetcli: ficheros de backup

La configuración propiamente dicha del servicio queda registrada en el directorio `/etc/target`, en el fichero `scsi_target.lio`

```
root@iscsiServer:/etc/target# ll
total 12
drwxr-xr-x 1 root root 30 abr 21 16:22 ./
drwxr-xr-x 1 root root 2970 abr 22 15:28 ../
-rw-r--r-- 1 root root 8834 may 2 21:20 scsi_target.lio
```

Figura 5.17. Targetcli: fichero de configuración

Dicho fichero guarda información sobre la ruta dentro del sistema de ficheros donde se encuentra el fichero real que exportamos como target, su tamaño y el número de serie (WWN) que el servicio le otorga, una lista de sus atributos e información sobre sus ACLs:

```
o- / ..... [..]
o- backstores ..... [..]
| o- fileio ..... [2 Storage Objects]
| | o- disk ..... [20.0G, /opt/pool/master/opensuse.img, in use]
| | o- ubuntu16 ..... [15.0G, /opt/pool/master/ubuntu16.img, in use]
| o- iblock ..... [0 Storage Object]
| o- pscsi ..... [0 Storage Object]
| o- rd_mcp ..... [0 Storage Object]
o- ib_srpt ..... [0 Targets]
o- iscsi ..... [2 Targets]
| o- iqn.2016-04.org.example:opensuse ..... [1 TPG]
| | o- tpg1 ..... [enabled]
| | | o- acs ..... [0 ACLs]
| | | o- luns ..... [1 LUN]
| | | | o- lun0 ..... [fileio/disk (/opt/pool/master/opensuse.img)]
| | o- portals ..... [1 Portal]
| | | o- 10.1.0.1:3260 ..... [OK, iser disabled]
| o- iqn.2016-04.org.example:ubuntu16 ..... [1 TPG]
| | o- tpg1 ..... [enabled]
| | | o- acs ..... [0 ACLs]
| | | o- luns ..... [1 LUN]
| | | | o- lun0 ..... [fileio/ubuntu16 (/opt/pool/master/ubuntu16.img)]
| | o- portals ..... [1 Portal]
| | | o- 10.1.0.1:3260 ..... [OK, iser disabled]
o- loopback ..... [0 Targets]
o- qla2xxx ..... [0 Targets]
o- tcm_fc ..... [0 Targets]
o- usb_gadget ..... [0 Targets]
o- vhost ..... [0 Targets]
```

Figura 5.18. Targetcli: targets

5.2. Instalación de Sistemas Operativos

5.2.1. Fedora

Primero, creamos un fichero, con un tamaño de 15 G y en la ruta /opt/pool/master, con el comando **truncate /opt/pool/master/disk.img -s 15G**

El siguiente paso es crear un Target para este fichero, que es donde instalaremos el SO, lo hacemos como hemos visto en el apartado A:

```
0- fileio ..... [3 Storage Objects]
  0- disk ..... [15.0G, /opt/pool/master/disk.img, in use]
0- iqn.2016-05.org.example:disk ..... [1 TPG]
  0- tpg1 ..... [enabled]
    0- acs ..... [0 ACLs]
    0- luns ..... [1 LUN]
      0- lun0 ..... [fileio/disk (/opt/pool/master/disk.img)]
    0- portals ..... [1 Portal]
      0- 10.1.0.1:3260 ..... [OK, iser disabled]
```

Figura 5.19. Targetcli: preparación para instalar un sistema operativo

La instalación se realizará desde un PC cliente donde insertaremos el disco de instalación de Fedora, este PC debe tener acceso al servidor iSCSI.

Iniciamos la instalación normalmente, como si fuéramos a instalar el SO en la máquina cliente; el paso más importante de este proceso llega cuando debemos escoger un disco duro donde instalar el SO: deberemos descubrir el Target iSCSI servido por nuestro servidor.

En este ejemplo, nuestra máquina cliente no dispone de un disco duro local, debemos seleccionar un disco iSCSI con la siguiente opción del instalador:

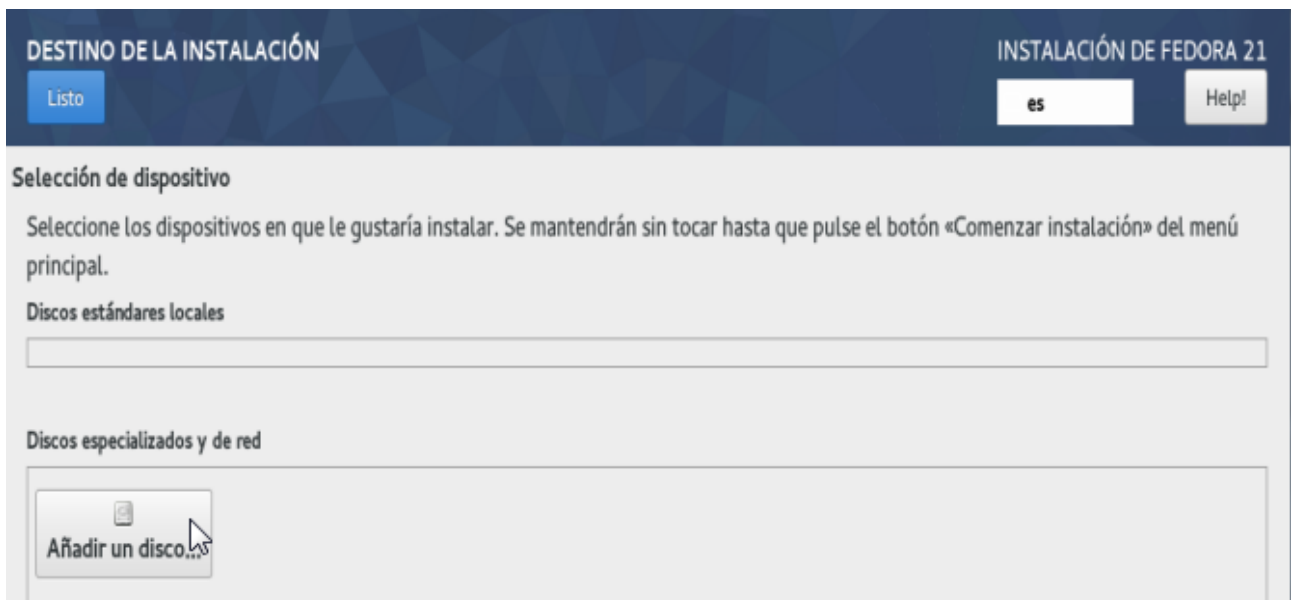


Figura 5.20. Instalación Fedora: selección de disco duro

En el siguiente paso, seleccionamos “Añadir Objetivo iSCSI”

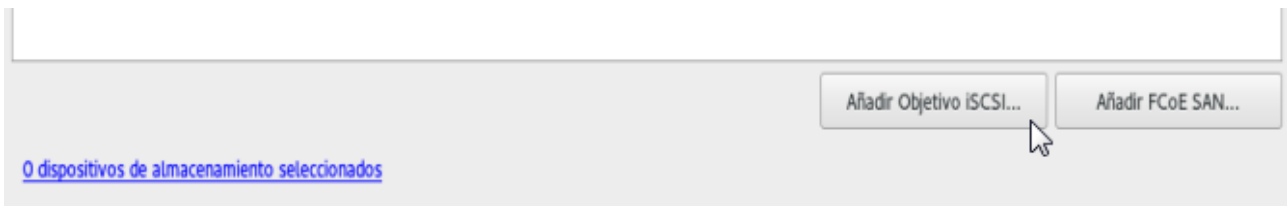


Figura 5.21. Instalación SO: añadir disco iSCSI

Indicamos la dirección IP del servidor iSCSI donde se encuentra el Target de destino, seleccionamos “Sin credenciales” (el Target no requiere Login) y hacemos click en “Iniciar Descubrimiento”:

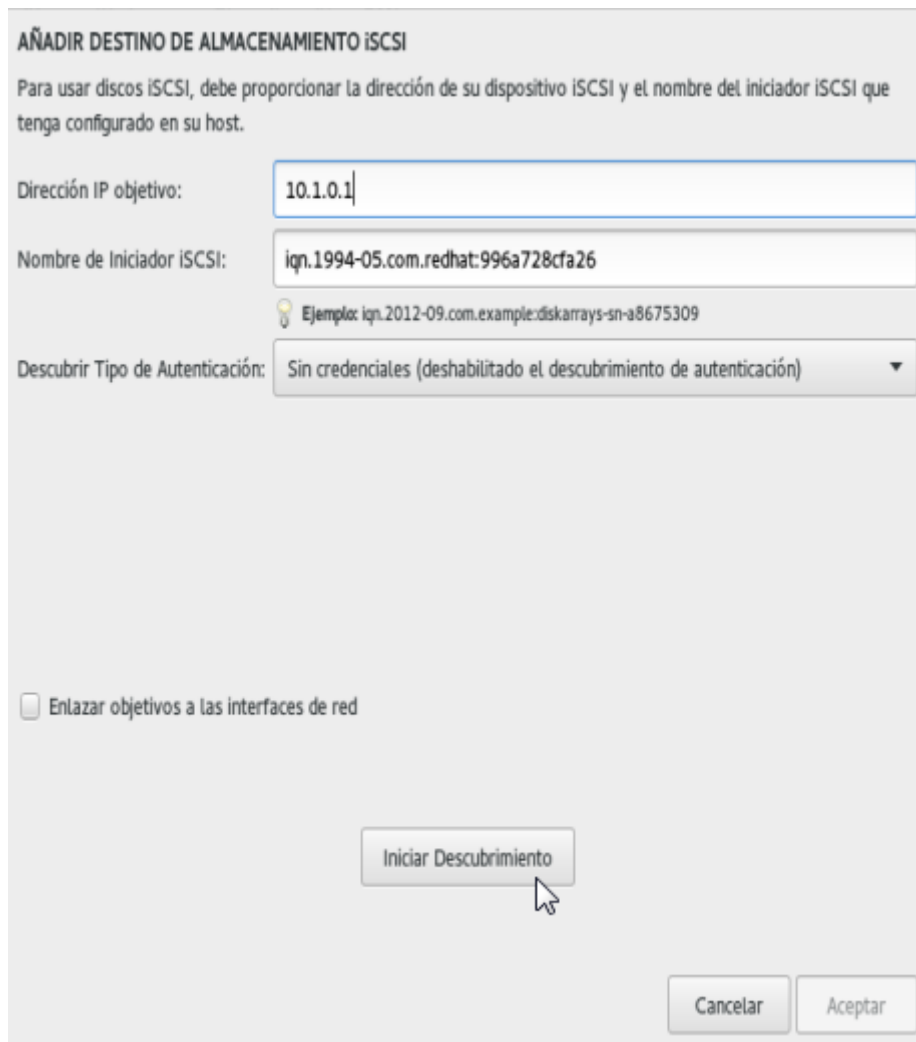


Figura 5.22. Instalación Fedora: descubrimiento Target iSCSI

El descubrimiento iSCSI muestra todos los Targets exportados desde el servidor iSCSI, seleccionamos el Target que hemos preparado para la instalación y hacemos click en “Acceder”

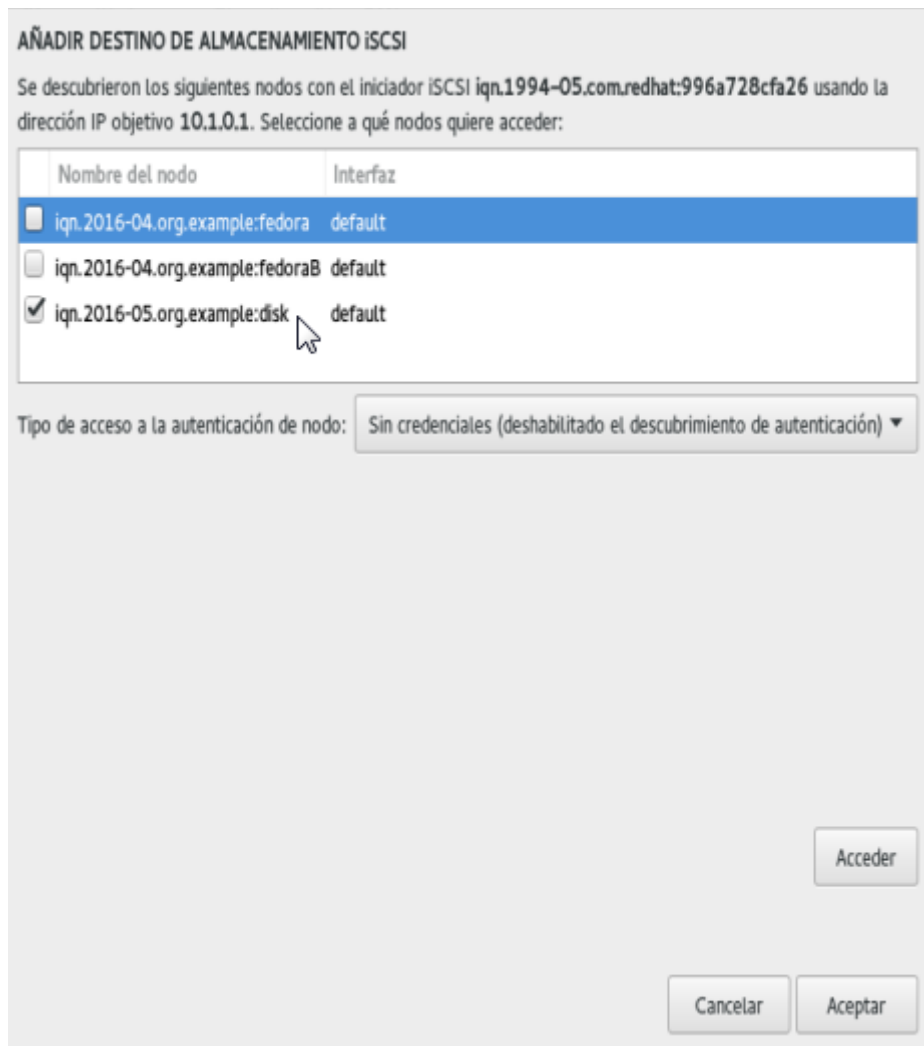


Figura 5.23. Instalación Fedora: selección de Target iSCSI

El disco iSCSI aparece en la lista de discos disponibles:



Figura 5.24. Instalación Fedora: Destino de la instalación



Figura 5.25. Instalación Fedora: Destino de la instalación

Llegados a este punto, conviene dar una breve explicación del porqué hemos elegido instalar la versión 21 de Fedora en lugar de la versión 23 (más reciente): en este punto del proceso de instalación, bastaría con hacer click sobre el botón “Listo” para proceder con la instalación, Fedora particionaría el disco de manera automática e instalaría GRUB en nuestro disco iSCSI. En la versión 23 esto no sucede: tanto el particionado automático como el manual devuelven un error al terminar, y la única solución que permite continuar con la instalación del SO pasa por no instalar GRUB en el disco iSCSI; esta opción es inviable, por lo que nos decantamos por la instalación de Fedora 21, que no presenta este problema.

A partir de este punto, la instalación del SO se realiza de manera convencional.

5.2.2 Ubuntu Server 16.04:

Tal como sucede con la instalación de Fedora, el punto clave de la instalación llega a la hora de seleccionar un disco duro donde instalar el SO (recordemos que los clientes virtuales que usamos en esta práctica no disponen de disco duro local)

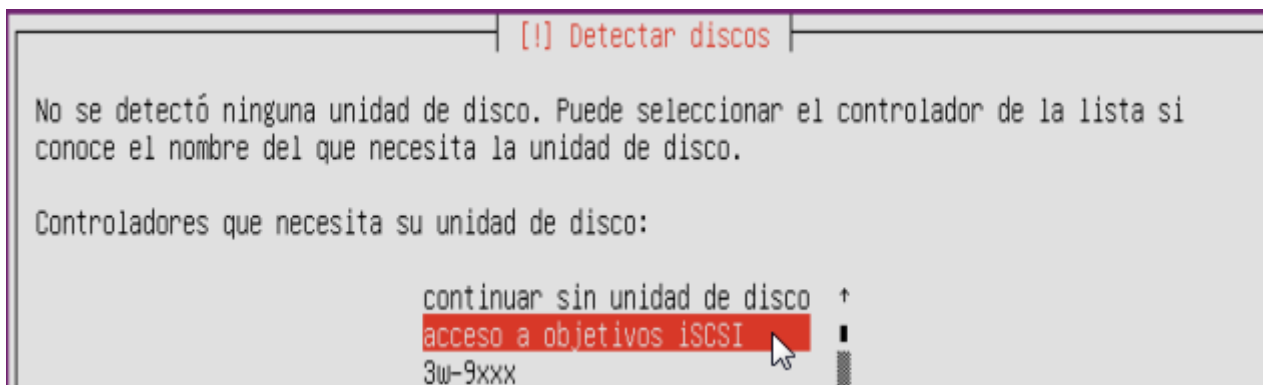


Figura 5.26. Instalación Ubuntu: Selección de disco

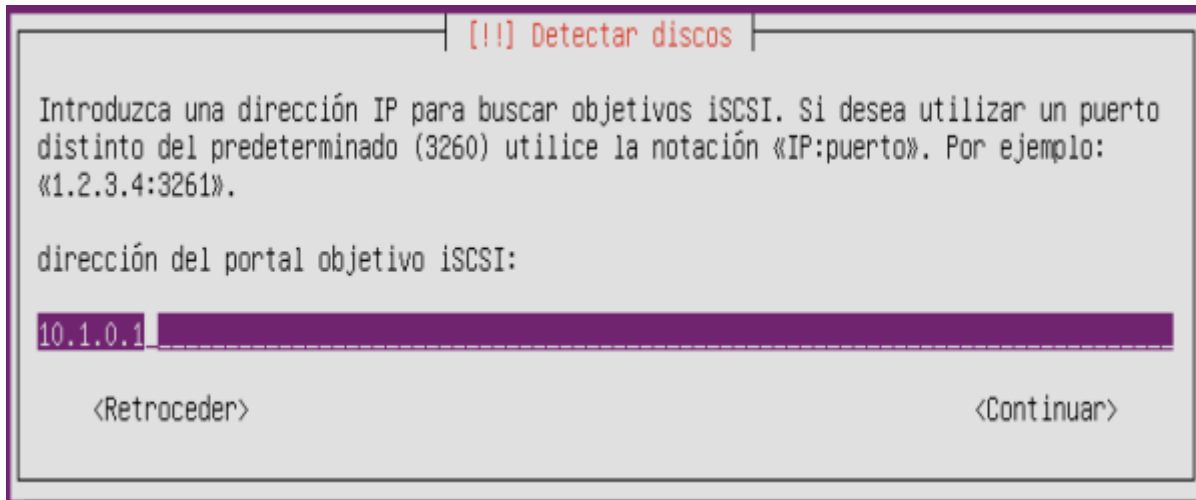
El instalador nos requiere cierta información para establecer la conexión con el Target iSCSI con el que queremos establecer conexión:

*Initiator Name: IQN (identificador único) para la máquina cliente; según los permisos de acceso que hayamos dispuesto en el Target, deberemos especificar un IQN concreto para el Initiator en este punto, de modo que sólo un determinado IQN Initiator pueda acceder al Target. Este no es el caso (cualquiera puede acceder al Target vacío para realizar una instalación; posteriormente, con los SSOO instalados en el servidor y los Targets exportados, sí que se establecerán políticas de login para acceder a los Targets). Dejamos esta información en blanco:



Figura 5.27. Instalación Ubuntu: nombre del iSCSI Initiator

Dirección IP del servidor iSCSI:



[!!] Detectar discos

Introduzca una dirección IP para buscar objetivos iSCSI. Si desea utilizar un puerto distinto del predeterminado (3260) utilice la notación «IP:puerto». Por ejemplo: «1.2.3.4:3261».

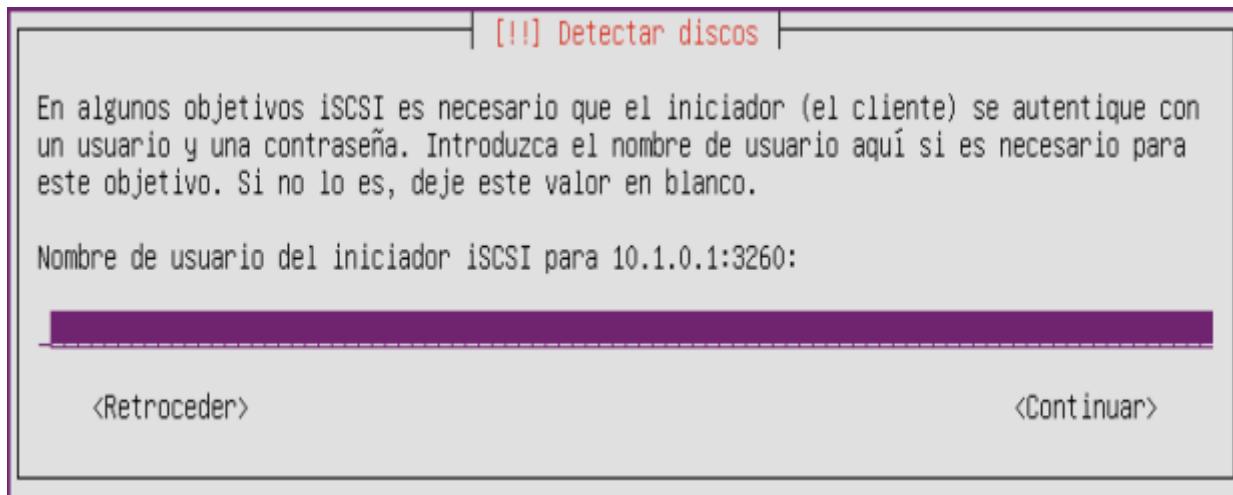
dirección del portal objetivo iSCSI:

10.1.0.1

<Retroceder> <Continuar>

Figura 5.28. Instalación Ubuntu: establecer dirección IP del servidor iSCSI

Login en el Target: no se necesita para acceder al Target a la hora de realizar la instalación, dejamos los campos en blanco:



[!!] Detectar discos

En algunos objetivos iSCSI es necesario que el iniciador (el cliente) se autentique con un usuario y una contraseña. Introduzca el nombre de usuario aquí si es necesario para este objetivo. Si no lo es, deje este valor en blanco.

Nombre de usuario del iniciador iSCSI para 10.1.0.1:3260:

<Retroceder> <Continuar>

Figura 5.29. Instalación Ubuntu: login en el iSCSI Target

El siguiente paso es seleccionar el Target adecuado para la instalación (“disk” es un fichero vacío, por lo que lo seleccionamos)



Figura 5.30. Instalación Ubuntu: selección de iSCSI Target

Pasado este punto, podemos continuar con la instalación normalmente, el dispositivo iSCSI responderá como si fuese un disco duro local.

5.3. Despliegue de imágenes clonadas

En un apartado posterior, veremos el script utilizado para automatizar la tarea de desplegar diferentes Targets a partir de una imagen maestra, pero antes conviene que nos detengamos en el proceso previo que debe realizarse para que el despliegue sea exitoso.

Cuando se establece una conexión por vez primera con un Target mediante el comando `iPXE sanboot`, el SO guarda en un fichero los parámetros necesarios para establecer la conexión en futuras ocasiones.

Si siempre conectamos al mismo Target (o generamos un Target nuevo para cada conexión), este hecho no supone un problema, pero lo que se pretende es, a partir de una imagen original (que no será utilizada por los clientes y que funcionará como un backup de los discos duros exportados) generar réplicas individuales de dicha imagen para los distintos usuarios.

Cuando el SO arranca, mira los parámetros de conexión que tiene localmente almacenados, y no los que le manda la BIOS (los parámetros especificados en el comando `iPXE sanboot iscsi:10.1.0.1:::iqn.YYYY-MM.dominio:target`), lo que resulta en que las conexiones al Target clonado acaban siendo en realidad conexiones al Target maestro.

Este problema tiene una fácil solución tanto en sistemas Ubuntu como Fedora.

En Ubuntu, debemos editar el fichero `/etc/iscsi/iscsi.initramfs`, que contiene la siguiente información:

```
root@ubuntu:/etc/iscsi# cat iscsi.initramfs
ISCSI_TARGET_NAME="iqn.2016-05.org.example:ubuntu"
ISCSI_TARGET_IP="10.1.0.1"
ISCSI_TARGET_PORT="3260"
ISCSI_TARGET_GROUP="1"
```

Figura 5.31. Archivo `iscsi.initramfs` en Ubuntu 16.04

Sustituimos el contenido del fichero por la siguiente línea:

```
ISCSI_AUTO=true
```

```

root@ubuntu:/etc/iscsi# echo "ISCSI_AUTO=true" > iscsi.initramfs
root@ubuntu:/etc/iscsi# cat iscsi.initramfs
ISCSI_AUTO=true

```

Figura 5.32. Archivo iscsi.initramfs en Ubuntu 16.04

El último paso es actualizar el fichero initramfs:

```

root@ubuntu:/etc/iscsi# update-initramfs -u
update-initramfs: Generating /boot/initrd.img-4.4.0-21-generic
W: plymouth: The plugin label.so is missing, the selected theme might not work as
s expected.
W: plymouth: You might want to install the plymouth-themes and plymouth-label pa
ckage to fix this.
W: mdadm: /etc/mdadm/mdadm.conf defines no arrays.

```

Figura 5.33. Actualización de initramfs

Realizados estos pasos, podemos proceder a la clonación de la imagen maestra y disponer así de una réplica lista para ser exportada; la copia la realizamos con la opción `--reflink="always"` propia del sistema de ficheros btrfs para crear una réplica que por el momento no ocupa espacio, y que sólo aumentará su tamaño en disco a medida que se diferencie del original.

```

root@iscsiServer:/opt/pool# cp --reflink="always" master/ubuntu.img images/ubuntu
A.img

```

Figura 5.34. Copia manual de un fichero de imagen

En Fedora el proceso a seguir es similar; primero necesitamos instalar el paquete `iscsi-initiator-utils` y ejecutar la siguiente orden:

```

[root@localhost user]# echo 'add_dracutmodules+=iscsi' > /etc/dracut.conf.d/ibft.conf

```

Figura 5.35. Modificación del fichero `ibft.conf` en Fedora

A continuación, editamos el fichero `/etc/default/grub` (en este fichero, el SO guarda la información referente al Target al que debe conectarse “por defecto”, omitiendo la información pasada por la BIOS en el momento del arranque por red (con el comando `SANBOOT`)) para añadir las siguientes entradas en la línea `GRUB_CMDLINE_LINUX`:

```

rd.iscsi.firmware=1
rd.iscsi.ibft=1
netroot=iscsi:ibft

```

Necesitamos actualizar GRUB con el comando `grub2-mkconfig -o /boot/grub2/grub.cfg`

```

[root@localhost user]# grub2-mkconfig -o /boot/grub2/grub.cfg
Generating grub configuration file ...
Found linux image: /boot/vmlinuz-3.17.4-301.fc21.x86_64
Found initrd image: /boot/initramfs-3.17.4-301.fc21.x86_64.img
Warning: Please don't use old title 'Fedora, with Linux 3.17.4-301.fc21.x86_64' for GRUB_DEFAULT, use 'Advanced options for Fe
dora>Fedora, with Linux 3.17.4-301.fc21.x86_64' (for versions before 2.00) or 'gnulinux-advanced-75f21879-56ae-4619-bef9-e7dc8
fadcee8>gnulinux-3.17.4-301.fc21.x86_64-advanced-75f21879-56ae-4619-bef9-e7dc8fadcee8' (for 2.00 or later)
Found linux image: /boot/vmlinuz-0-rescue-e9218bd702ae43cdab208347a20c7fe0
Found initrd image: /boot/initramfs-0-rescue-e9218bd702ae43cdab208347a20c7fe0.img
done

```

Figura 5.36. Actualización de GRUB en Fedora 21

5.4. Permisos de acceso a los Targets

Aunque LIO permite la implantación de ACLs para permitir o denegar el acceso a un Target por parte de determinados Initiators (de manera que un mismo Target pueda ser accedido por unos determinados clientes en exclusiva e identificados por sus IQNs), nos hemos decantado por una implementación de permisos más flexible, pues el método de discriminar por IQN de los Initiators supondría que siempre deberíamos iniciar sesión con el Target desde una misma máquina cliente.

Para cada Target exportado especificaremos un método de login, de manera que sólo los usuarios autorizados puedan acceder a dicho Target, independientemente del Initiator IQN que tengan.

Los comandos para establecer permisos de acceso para los Targets son:

```

set attribute authentication=1
demo_mode_write_protect=0
generate_node_acls=1
cache_dynamic_node_acls=1

set auth userid = username
set auth password= password

```

```

/iscsi> cd iqn.2016-04.org.example:fedoraB
/iscsi/iqn.20...ample:fedoraB> cd tpg1/
/iscsi/iqn.20...:fedoraB/tpg1> set attribute authentication=1 demo_mode_write_protect=0 generate_node_acls=1 cache_dynamic_acls=1
Parameter demo_mode_write_protect is now '0'.
Parameter authentication is now '1'.
Parameter generate_node_acls is now '1'.
Parameter cache_dynamic_acls is now '1'.
/iscsi/iqn.20...:fedoraB/tpg1> set auth userid=user
Parameter userid is now 'user'.
/iscsi/iqn.20...:fedoraB/tpg1> set auth password=123
Parameter password is now '123'.

```

Figura 5.37. Targetcli: configuración de permisos de acceso para un Target

Comprobamos el fichero que guarda la configuración de los Targets creados con Targetcli: `/etc/target/scsi_target.lio` para ver que efectivamente los permisos están activados:

```

target iqn.2016-04.org.example:fedoraB tpgt 1 {
  enable yes
  attribute {
    authentication yes
    cache_dynamic_acls yes
    default_cmdsndepth 64
    default_erl 0
    demo_mode_discovery yes
    demo_mode_write_protect no
    fabric_prot_type 0
    generate_node_acls yes
    login_timeout 15
    netif_timeout 2
    prod_mode_write_protect no
    t10_pi 0
    tpg_enabled_sendtargets 1
  }
  auth {
    password 123
    password_mutual ""
    userid user
    userid_mutual ""
  }
  parameter {
    AuthMethod CHAP
  }
}

```

Figura 5.38. Targetcli: fichero de configuración

En el menú de arranque, tan sólo resta editar la línea que permite el arranque del Target para añadir como primera orden el comando iPXE *login* (al pedir acceso al target con ACLs, lo primero que verá el cliente será una pantalla de login donde deberá introducir nombre de usuario y contraseña válidos). Si el login es exitoso, comenzará el proceso de arranque normal del Target iSCSI (se ejecutará el comando iPXE *sanboot*)

```

:fedoraB
login && sanboot iscsi:10.1.0.1:::iqn.2016-04.org.example:fedoraB

```

Figura 5.39. Ejemplo de script de menú iPXE con login



Figura 5.40. iPXE: pantalla de login

6. Creación de scripts

Una vez analizado y probado el funcionamiento del proceso de creación y publicación de imágenes en portales iSCSI, implementamos unos scripts que automatizan todas las tareas que hemos estado realizando.

Hemos organizado el despliegue con scripts básicos para gestionar los targets iSCSI de la forma más simple posible, sin perder la versatilidad a la hora del despliegue. Ahora indicaremos los procesos y los conceptos utilizados.

6.1. Conceptos targetmgr

Hemos definido dos términos diferenciados para una imagen:

- **Master:** Imagen fileio con un sistema operativo ya preparado para su despliegue, (ver apartado 6.2) que se almacenará en el directorio especificado en el valor de la ruta master indicado en el fichero de configuración *targetmgr.conf*.
 - Ejemplo ruta master: */opt/pool/master/ubuntu.img*
- **Imagen:** Imagen clonada de un master u otra imagen. Este fichero se guardará en la ruta de ficheros de imagenes indicada en el fichero *targetmgr.conf* añadiendo los subdirectorios indicados en el parámetro (*naming_authority -n*) del comando *tgtaddtarget*.
 - Ejemplo ruta imagen: */opt/pool/images/dam1/m01/uf1/ubuntu_alumno1.img*

NOTA: La extensión final siempre es *.img*, los scripts de gestión de targets lo crean automáticamente y son necesarios en la generación de los menus iPXE al igual que el guión bajo que concatena el master con el usuario en la imagen desplegada.

Referencia a las operaciones a realizar:

- **Desplegar:** publicar, hacer accesible el target ... creando el fileio ..bla bla
- **Replegar;** eliminar el target y eliminar la imagen del sistema de fichero....bla bla

Con referencia a los *roles* en cuanto a la utilización de los scripts, hemos definido los siguientes.

- **Usuario:** Por motivos de simplicidad hemos descartado la creación de grupos y tanto alumnos como profesores, utilizan el mismo rol. Como valor único identificativo, sugerimos utilizar el nombre de correo electrónico del centro.
- **Master:** Usuario '*master*' utilizado en la creación de imagenes *master*. Necesario para autenticarse en el menú iPXE y poder arrancar las imagenes master.

NOTA: Utilizar el despliegue creando directorios equivalentes al *naming_authority* permite ordenar las imagenes con cualquier jerarquia imaginable, por ejemplo */curso/asignatura/modulo/unidad o /departamento/profesor/asignatura*.

6.2. Directorio */etc/targetmgr*

Dentro de este directorio encontramos el fichero de configuración utilizado por los scripts de despliegue y repliegue para realizar las operaciones, junto con el fichero de usuarios que contiene las correspondientes contraseñas.

6.2.1. Fichero *targetmgr.conf*

El fichero de configuración incluye todos los valores que utilizarán los scripts para desplegar las imágenes. Los comentarios indican el uso de cada valor.

```
#-----  
# targetmgr.conf  
# Configuración para comandos target y menú iPXE  
#  
# IQN_Format:  
# # iqn.yyyy-mm.naming-authority:unique_name  
#  
#-----  
  
[SERVER]  
# Datos del servidor  
server=192.168.1.100  
port=3260  
  
[PATH]  
# Rutas de las imágenes  
path_master=/opt/pool/master/  
path_images=/opt/pool/images/  
path_log=/etc/targetmgr/  
  
[MASTER]  
# Contraseña usuario master  
pwd_master=secreto  
# iqn naming-authority  
iqn_naming_master=org.puig.master  
  
[IQN]  
# format iqn.yyy-mm  
iqn_prefix=iqn.2016-04
```

Figura 6.1. Fichero de configuración

6.2.2. Fichero *users*

Se ha definido un fichero con la nomenclatura *usuario:password* donde se almacenan todas las claves de acceso de los usuarios: (ver roles apartado 6.1).

```
alumno1:111  
alumno2:222  
alumno3:333  
profesor1:aaa  
profesor2:bbb
```

Figura 6.2. Fichero *users*

NOTA: Este par de valores se utilizarán, tanto en el acceso a los targets del servidor, como en la autorización del menú iPXE.

6.3. Scripts de *targetmgr*

Pasamos a explicar los diferentes scripts con diferentes casos de uso. Empezamos con los scripts de creación y clonación de un master (*tgtcreatemaster* y *tgtclonemaster*), después veremos la clonación a una imagen (*tgtcloneimage*). Para terminar con el repliegue de las imágenes (*tgtdeleteimage*) y de los masters (*tgtdeletemaster*).

6.3.1. Script *tgtcreatemaster*

Utilizamos este script para crear un *backstores/fileio* del master y preparar el target, sus valores de autenticación serán *master/pwd_master* del fichero de configuración visto anteriormente.

```
ISCSI Target Manager - tgtcreatemaster
tgtcreatemaster. Utilizar para crear masters y desplegar en iscsi
Opciones:
  -m master      Nombre del master a crear.
                  - No incluir puntos (.) ni subrayados (_)
  -s size        Tamaño del fileio del master.
  -h             Muestra la ayuda
Ejemplo:
  tgtcreatemaster -m UbuntuServer16-04 -s 10G
```

Figura 6.3. Script *tgtcreatemaster*: parámetros

RECORDAR: El *master* todavía no contiene el sistema operativo, es necesario seguir los pasos del apartado 6.2, y ahora introducir las credenciales de validación.

Caso de uso: crear un master para instalar UbuntuServer16.04

Vamos a preparar un master para instalar ubuntu Server 16.04, una vez creado es necesario instalar el sistema operativo, quedando operativo para su posterior clonación a un master o imagen.

```
# tgtcreatemaster -m UbuntuServer16-04 -s 10G
```

Veamos la entrada del fichero log:

```
[CREATEMASTER]
iscsi:   iqn.2016-04.org.puig.master:UbuntuServer16-04
fileio:  _UbuntuServer16-04
file:    UbuntuServer16-04.img
```

NOTA: Se puede observar que se ha añadido un subrayado al nombre del */backstores/fileio* para que los masters siempre aparezcan al principio de la lista.

6.3.2 Script *tgtclonemaster*

Utilizado para clonar un master ya operativo y añadirle nuevas funcionalidades, por ejemplo añadir un escritorio o un entorno de desarrollo. La imagen del fichero se guarda en la ruta master. Se realiza una copia-on-write del master para aprovechar espacio de disco.

```

ISCSI Target Manager - tgtclonemaster

tgtclonemaster. Utilizado para clonar un master a otro master

Opciones:

  -m master      Nombre del master a clonar
  -c clone       Nombre del clone
                  - No incluir puntos ni subrayados

  -h             Muestra la ayuda

Ejemplo:

tgtclonemaster -m master -c clone

```

Figura 6.4. Script *tgtclonemaster*: parámetros

Caso de uso: Clonar UbuntuServer16 para añadir entorno de escritorio

Clonamos un master ya operativo de *UbuntuServer* a un nuevo master *UbuntuDesktop*. Una vez desplegado arrancamos por iPXE el nuevo *UbuntuDesktop* e instalamos el entorno de escritorio y lo dejamos preparado para su posterior despliegue.

```
# tgtcreatemaster -m UbuntuServer16-04 -c UbuntuDesktop16-4
```

Veamos el fichero log generado:

```

[CLONEMASTER]
iscsi:      iqn.2016-04.org.puig.master:UbuntuDesktop16-04
fileio:     _UbuntuDesktop16-04
file:       UbuntuDesktop16-04.img
From master: UbuntuServer16-04.img

```

6.3.3. Script *tgtcloneimage*

Una vez preparado el master para su clonación necesitamos clonar la imagen, y desplegar el target para poder acceder desde el menú iPXE mediante la correspondiente autenticación del usuario. Las opciones del script son las siguientes.

```

ISCSI Target Manager - tgtcloneimage

tgtcloneimage. Utilizar para clonar un master o imagen y desplegar el target.

Opciones:

  [- m master|-i image] [-u user|-f file_users] [-n naming-authority]

  -m master      Nombre del master a clonar.

  -i image       naming-authority y unique name de la imagen a clonar

  -u usuario     Usuario asignado al target a desplegar

  -f file_users  Fichero de usuario para despliegue por lotes

  -n naming-auth Naming-authority del target a desplegar

  -h             Muestra la ayuda

Ejemplos:

tgtcloneimage -m master -u usuario -n xx.yy.zz
tgtcloneimage -i xx.yy.zz:usuario -f usuarios -n xx.yy.zz

```

Figura 6.5. Script *tgtcloneimage*: parámetros

Casos de uso: Clonar una única imagen

Un profesor clona una imagen master con el objetivo de instalar diferentes aplicaciones y preparar el sistema operativo para realizar una actividad de su asignatura.

```
#tgtcloneimage -m UbuntuDesktop16-04 -u profesor1 -n dept.activ
```

Veamos el fichero log:

```
[CLONEIMAGE]
iscsi:   iqn.2016-04.dept.activ.UbuntuDesktop16-04:profesor1
fileio:  dept.activ.UbuntuDesktop16-04:profesor1
from:    /opt/pool/master/UbuntuDesktop16-04.img
to:      /opt/pool/images/dept/activ/UbuntuDesktop16-04_profesor1.img
```

Casos de uso: Clonar múltiples imágenes

Ahora tenemos la posibilidad de clonar la imagen anterior a más de un usuario, mediante un fichero de usuarios que puede representar un grupo o clase particular. Creamos un fichero *usuarios* que contiene los usuarios *alumno2* y *alumno3*.

```
# tgtcloneimage -i dept.activ.UbuntuDesktop-04:profesor -f usuarios -n asix2.m01.uf1
```

Veamos el fichero log

```
[CLONEIMAGE+]
iscsi:   iqn.2016-04.asix2.m01.uf1.UbuntuDesktop16-04:alumno2
fileio:  asix2.m01.uf1.UbuntuDesktop16-04:alumno2
from:    /opt/pool/images/dept/activ/UbuntuDesktop16-04_profesor1.img
to:      /opt/pool/images/asix2/m01/uf1/UbuntuDesktop16-04_alumno2.img

[CLONEIMAGE+]
iscsi:   iqn.2016-04.asix2.m01.uf1.UbuntuDesktop16-04:alumno3
fileio:  asix2.m01.uf1.UbuntuDesktop16-04:alumno3
from:    /opt/pool/images/dept/activ/UbuntuDesktop16-04_profesor1.img
to:      /opt/pool/images/asix2/m01/uf1/UbuntuDesktop16-04_alumno3.img
```

En la figura siguiente podemos ver como queda la estructura de los targets entrando con la utilidad `targetcli`:

```

/> ls
0- / ..... [1]
0- backstores ..... [6 Storage Objects]
0- fileio ..... [10.0G, /opt/pool/master/UbuntuDesktop16-04.img, in use]
0- UbuntuDesktop16-04 ..... [10.0G, /opt/pool/master/UbuntuDesktop16-04.img, in use]
0- UbuntuServer16-04 ..... [10.0G, /opt/pool/master/UbuntuServer16-04.img, in use]
0- asix2.m01.uf1.UbuntuDesktop16-04:alumno2 ... [10.0G, /opt/pool/images/asix2/m01/uf1/UbuntuDesktop16-04_alumno2.img, in use]
0- asix2.m01.uf1.UbuntuDesktop16-04:alumno3 ... [10.0G, /opt/pool/images/asix2/m01/uf1/UbuntuDesktop16-04_alumno3.img, in use]
0- asix2.m02.uf2.UbuntuServer16-04:alumno3 . [10.0G, /opt/pool/images/asix2/m02/uf2/UbuntuServer16-04_alumno3.img, not in use]
0- dept.activ.UbuntuDesktop16-04:profesor1 ..... [10.0G, /opt/pool/images/dept/activ/UbuntuDesktop16-04_profesor1.img, in use]
0- iblock ..... [0 Storage Object]
0- pscsi ..... [0 Storage Object]
0- rd_mcp ..... [0 Storage Object]
0- ib_srpt ..... [0 Targets]
0- iscsi ..... [5 Targets]
0- iqn.2016-04.asix2.m01.uf1.UbuntuDesktop16-04:alumno2 ..... [1 TPG]
0- tpg1 ..... [enabled]
0- acls ..... [0 ACLs]
0- luns ..... [1 LUN]
0- lun0 [fileio/asix2.m01.uf1.UbuntuDesktop16-04:alumno2 (/opt/pool/images/asix2/m01/uf1/UbuntuDesktop16-04_alumno2.img)]
0- portals ..... [1 Portal]
0- 192.168.1.100:3260 ..... [OK, user disabled]
0- iqn.2016-04.asix2.m01.uf1.UbuntuDesktop16-04:alumno3 ..... [1 TPG]
0- tpg1 ..... [enabled]
0- acls ..... [0 ACLs]
0- luns ..... [1 LUN]
0- lun0 [fileio/asix2.m01.uf1.UbuntuDesktop16-04:alumno3 (/opt/pool/images/asix2/m01/uf1/UbuntuDesktop16-04_alumno3.img)]
0- portals ..... [1 Portal]
0- 192.168.1.100:3260 ..... [OK, user disabled]
0- iqn.2016-04.dept.activ.UbuntuDesktop16-04:profesor1 ..... [1 TPG]
0- tpg1 ..... [enabled]
0- acls ..... [0 ACLs]
0- luns ..... [1 LUN]
0- lun0 [fileio/dept.activ.UbuntuDesktop16-04:profesor1 (/opt/pool/images/dept/activ/UbuntuDesktop16-04_profesor1.img)]
0- portals ..... [1 Portal]
0- 192.168.1.100:3260 ..... [OK, user disabled]
0- iqn.2016-04.org.puig.master:UbuntuDesktop16-04 ..... [1 TPG]
0- tpg1 ..... [enabled]
0- acls ..... [0 ACLs]
0- luns ..... [1 LUN]
0- lun0 [fileio/UbuntuDesktop16-04 (/opt/pool/master/UbuntuDesktop16-04.img)]
0- portals ..... [1 Portal]
0- 192.168.1.100:3260 ..... [OK, user disabled]
0- iqn.2016-04.org.puig.master:UbuntuServer16-04 ..... [1 TPG]
0- tpg1 ..... [enabled]
0- acls ..... [0 ACLs]
0- luns ..... [1 LUN]
0- lun0 [fileio/UbuntuServer16-04 (/opt/pool/master/UbuntuServer16-04.img)]
0- portals ..... [1 Portal]
0- 192.168.1.100:3260 ..... [OK, user disabled]
0- loopback ..... [0 Targets]
0- qla2xxx ..... [0 Targets]
0- tcm_fc ..... [0 Targets]
0- usb_gadget ..... [0 Targets]
0- vhost ..... [0 Targets]
/>

```

Figura 6.6. Targetcli: estructura de los Targets

Y la estructura de directorios generada:

```

root@iscsiServer:/home/user/targetmgr# tree /opt/pool
/opt/pool
├── images
│   ├── asix2
│   │   ├── m01
│   │   │   ├── uf1
│   │   │   │   ├── UbuntuDesktop16-04_alumno2.img
│   │   │   │   └── UbuntuDesktop16-04_alumno3.img
│   │   └── dept
│   │       └── activ
│   │           └── UbuntuDesktop16-04_profesor1.img
│   └── master
│       ├── UbuntuDesktop16-04.img
│       └── UbuntuServer16-04.img

```

Figura 6.7. Estructura de directorios

6.3.4 Script *tgtdeletemaster*

Cuando dejamos de utilizar una imagen master, utilizaremos este script para replegar su iSCSI target y su fileio asociado al LUN. Por defecto elimina también el fichero imagen del sistema de archivos, hemos activado la opción de crear una copia de seguridad del fichero de imagen, añadiendo una extensión de tiempo.

```
ISCSI Target Manager - tgtdeletemaster
tgtdeletemaster. Utilizar para replegar target iSCSI masters y
eliminar su imagen asociada del sistema de archivos
Opciones:
  -m master      Nombre de la imagen master a replegar
                  - No incluir extensión .img
  -b             Crear un backup de la imagen master.
  -h             Muestra la ayuda
Ejemplo:
  tgtdeletemaster -m master -b
```

Figura 6.8. Ayuda script *tgtdeletemasater*

Casos de uso: Eliminar un master

Ha sido lanzada una nueva versión del sistema operativo y deseamos eliminar el sistema operativo anterior. Procedemos a borrar la imagen master y su target iSCSI asociado.

```
#tgtdeletemaster -m UbuntuDesktop16-04
```

Veamos el fichero log:

```
[DELETEMASATER]
iscsi: iqn.2016-04.org.puig.master:UbuntuDesktop16-04
fileio: _ubuntu
file: UbuntuDesktop16-04.img
```

Casos de uso: Eliminar un master

Ha sido lanzada una nueva versión del sistema operativo y deseamos eliminar el sistema operativo anterior. Procedemos a borrar la imagen master y su target iSCSI asociado.

```
#tgtdeletemaster -m UbuntuDesktop16-04
```

Veamos el fichero log:

```
[DELETEMASATER]
iscsi: iqn.2016-04.org.puig.master:UbuntuDesktop16-04
fileio: _ubuntu
file: UbuntuDesktop16-04.img
```

6.3.5. Script *tgtdeleteimage*

Cuando ya no utilizamos una imagen clonada, utilizaremos este script para replegar su iSCSI target y su fileio asociado al LUN. Al igual que el script anterior, es posible activar la opción de backup del fichero de imagen. Para indicar las imágenes a borrar es posible indicar parte de *naming-authority* y el script se encarga de replegar todos los iSCSI de la jerarquía. También es posible borrar una sola imagen indicando su *naming-authority* más su *unique_name*.

```
ISCSI Target Manager - tgtdeleteimage

tgtdeleteimage. Utilizar para replegar imagenes target iSCSI y
eliminar su imagen asociada del sistema de archivos
Opciones:

  -i name      naming-authority a eliminar imagenes recursivamente o
               naming-authority y unique name para eliminar una imagen

  -b          Crear un backup de la imagen master.

  -t          Modo test. Visualiza imagenes sin eliminarlas

  -h          Muestra la ayuda

Ejemplo:

tgtdeleteimage -i xx.yy -b
tgtdeleteimage -i xx.yy.zz:unique_name -b
```

Figura 6.9. Ayuda script *tgtdeleteimage*

Casos de uso: Eliminar una unica imagen

Los alumnos han terminado una actividad y el profesor desea eliminar la imagen de clonación para liberar espacio en el sistema de archivos. Procedemos a borrar la imagen y su target iSCSI asociado.

```
#tgtdeleteimage -i dept.activ.UbuntuDesktop-04:profesor1
```

Veamos el fichero log:

```
[DELETEIMAGE]
iscsi: iqn.2016-04.dept.activ.UbuntuDesktop16-04:profesor1
fileio: dept.activ.UbuntuDesktop16-04:profesor1
file: /opt/pool/images/dept/activ/UbuntuDesktop16-04_profesor1.img
```

Casos de uso: Eliminar múltiples imagenes

Los alumnos ha terminado un modulo formativo y el profesor desea eliminar las imágenes de los alumnos para liberar espacio en el sistema de archivos. Procedemos a borrar las imágenes y sus targets iSCSI asociados.

```
#tgtdeleteimage -i asix2.m01
```

Veamos el fichero log:

```
[DELETEIMAGE]
iscsi: iqn.2016-04.asix2.m01.uf1.UbuntuDesktop16-04:alumno3
fileio: asix2.m01.uf1.UbuntuDesktop16-04:alumno3
file: /opt/pool/images/asix2/m01/uf1/UbuntuDesktop16-04_alumno3.img
[DELETEIMAGE]
iscsi: iqn.2016-04.asix2.m01.uf1.UbuntuDesktop16-04:alumno2
fileio: asix2.m01.uf1.UbuntuDesktop16-04:alumno2
file: /opt/pool/images/asix2/m01/uf1/UbuntuDesktop16-04_alumno2.img
```

7. Menu iPXE

Una vez desplegados todos los targets es necesario crear un menú en el arranque por iPXE, para que los usuarios accedan a sus imágenes, al igual que el usuario master, que también necesita acceder a sus imágenes masters.

Para implementar esta funcionalidad, hemos utilizado el lenguaje PHP que generará el correspondiente script iPXE, que se encargará de gestionar los menús de arranque. Por lo tanto, creamos los scripts en la carpeta `/var/www/html/`

NOTA: Para soportar comandos iPXE, tales como *params* que permite el paso de parámetros entre scripts PHP, y el comando *console* para activar la resolución de la pantalla y permitir imágenes PNG, es necesario descargar el proyecto ipxe y compilarlo de nuevo definiendo algunas directivas y actualizar el fichero *undionly.kpe* en el servidor ftp.

Por motivos de seguridad hemos creado una autenticación inicial para ver solamente las imágenes del usuario autenticado.

La primera pantalla del menú iPXE nos pide autenticarnos o continuar con el arranque local, hemos dejado esta opción por defecto con un temporizador de 10 segundos.

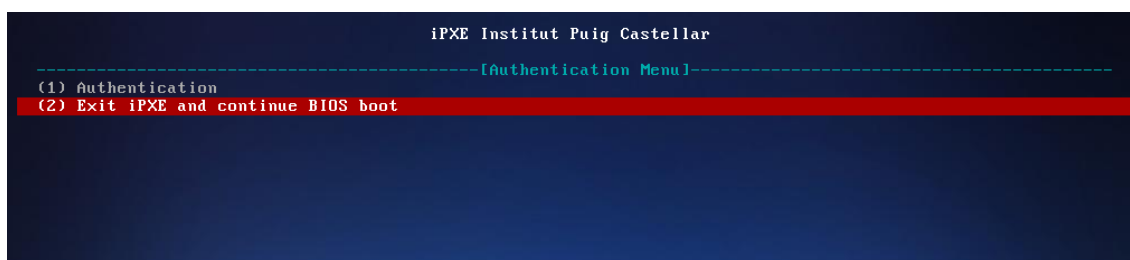


Figura 7.1. Pantalla autenticación iPXE

Si elegimos *Authentication*, y las credenciales introducidas son correctas, previa comprobación del fichero *users* del directorio `/etc/targetmgr`, entraremos en el menú de selección de imágenes.

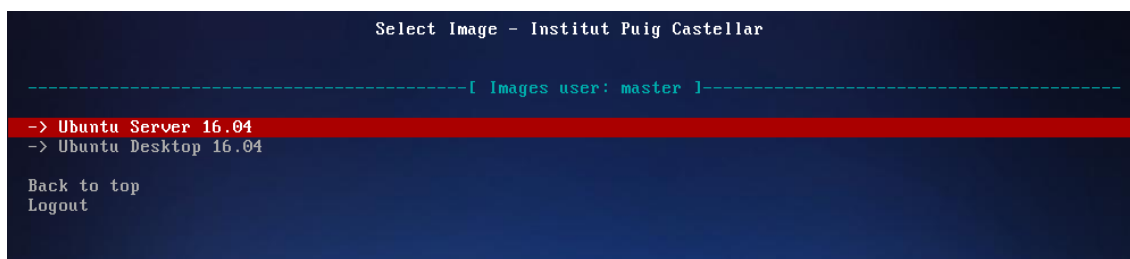


Figura 7.2. Selección de imágenes publicadas para master

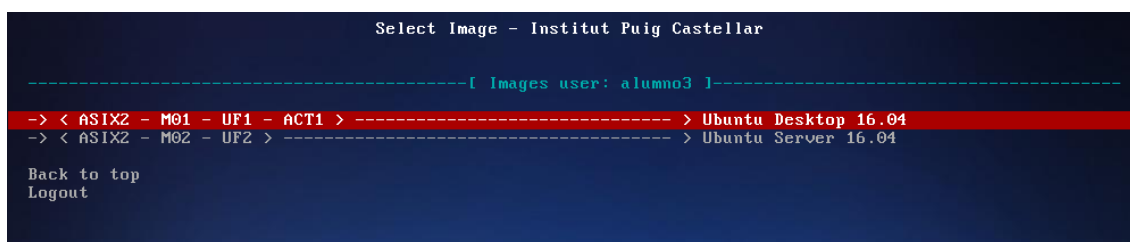


Figura 7.3. Selección de imágenes publicadas para alumno3

NOTA: La generación de las instrucciones sanboot y de los elementos de menús se recuperan a través de la estructura de directorio y de los datos del fichero de configuración de *targetmgr.conf*.

7.1. Comandos iPXE generados.

Los scripts php se encargan de generar los comandos iPXE que crean los menús que hemos visto en el apartado anterior. También realizan la validación de los usuarios, comprobando el fichero *users*, y del usuario *master*, comprobando el valor *pwd_master* del fichero de configuración.

Para generar las entradas y comandos de menús, los scripts realizan una búsqueda en el directorio de imágenes de ficheros con el nombre del usuario autenticado y recogen información del fichero de configuración para rellenar los comandos.

boot.php

Este script genera el menú inicial de autenticación y pasa los datos al script *menu.php* que realiza las tareas de validación y generación de menús.

```
#!/ipxe
console --x 1024 --y 768 --picture http://192.168.1.100:80/fondo.png
:menu
menu iPXE Institut Puig Castellar
set menu-default exit
set menu-timeout 10000
item --gap -- -----[ Authentication Menu ]-----
item --key 1 login (1) Authentication
item --key 2 exit (2) Exit iPXE and continue BIOS boot
choose --default ${menu-default} --timeout ${menu-timeout} selected && goto ${selected}
|| exit 0
:login
login
isset ${username} && isset ${password} || goto menu
params
param username ${username}
param password ${password}
chain --replace --autofree http://192.168.1.100:80/menu.php##params
:exit
echo Booting from local disks ...
exit 0
```

menu.php

Este script genera los menús para el usuario autenticado. Mostramos los comandos iPXE para *master* y para un usuario. Utilizamos el comando *wget* pasando los valores por POST.

```
wget http://192.168.1.100/menu.php --post-data="username=master&password=secreto"
#!/ipxe
params
param username ${username}
param password ${password}
:menu
menu Select Image - Institut Puig Castellar
item --gap
item --gap -- -----[ Images user: master ]-----
item --gap
item n1 -> Ubuntu Server 16.04
item n2 -> Ubuntu Desktop 16.04
item --gap
item backtotop Back to top
item signin Logout
choose selected && goto ${selected} || exit 0
:n1
sanboot iscsi:192.168.1.100::3260::iqn.2016-04.org.puig.master:UbuntuServer16-04
:n2
sanboot iscsi:192.168.1.100::3260::iqn.2016-04.org.puig.master:UbuntuDesktop16-04
:backtotop
goto menu
:signin
chain --replace --autofree http://192.168.1.100:80/boot.php
```

```

wget http://192.168.1.100/menu.php --post-data="username=alumne3&password=333"
#!ipxe
params
param username ${username}
param password ${password}
:menu
menu Select Image - Institut Puig Castellar
item --gap
item --gap -- -----[ Images user: alumno3 ]-----
item --gap
item n1 -> ASIX2 - M01 - UF1 - ACT1 > ----- > Ubuntu Desktop 16.04
item n2 -> ASIX2 - M02 - UF2 > ----- > Ubuntu Server 16.04
item --gap
item backtotop Back to top
item signin Logout
choose selected && goto ${selected} || exit 0
:n1
sanboot iscsi:192.168.1.100::3260::iqn.2016-04.asix2.m01.uf1.act1.UbuntuDesktop16-
04:alumno3
:n2
sanboot iscsi:192.168.1.100::3260::iqn.2016-04.asix2.m02.uf2.UbuntuServer16-04:alumno3
:backtotop
goto menu
:signin
chain --replace --autofree http://192.168.1.100:80/boot.php

```

8. Conclusiones

Para terminar describiremos las conclusiones que hemos sacado durante la realización del proyecto.

- Finalizado todo el proceso de configuración e implementación del servicio, tenemos un servidor SAN iSCSI que cumple con sus tareas principales.
- Hemos aprendido a poner en marcha diferentes servicios de red aprendidos durante el ciclo, como DHCP, TFTP.
- Profundizado en la tecnología iPXE y puesto en práctica de los comandos que incorpora.
- Hemos aprendido las funcionalidades de target iSCSI sobre Linux, y como gestionar el programa targetcli que los gestiona.
- También hemos analizado la API del módulo rstlib e implementado los scripts de python necesarios, nos ha servido de ayuda las clases de scripts para sistemas operativos.
- Hemos realizado scripts en PHP que también hemos aprendido durante el ciclo.

Objetivos no conseguidos:

- No hemos podido realizar pruebas intensas con más de cinco usuarios.
- Hemos conseguido clonar imagenes con ZFS pero por falta de tiempo no han sido incluido en los scripts de gestión de targets. Incluimos anexo con las operaciones a realizar en ZFS.
- No hemos podido realizar instalaciones de otros sistemas operativos, aunque lo hemos intentado, como openSUSE y ReactOS.

Mejoras y funcionalidades a añadir:

- Scripts de monitorización de los usuarios conectados al servidor iSCSI.
- Software por web, que gestione las operaciones sobre los targets iSCSI, junto a un interface para la gestión de los usuarios, y si es posible la monitorización en vivo de los usuarios conectados. Obteniedo un paquete completo con todas las funcionalidades.
- Por motivos de tiempo durante el proyecto, no hemos implemenado las contraseñas con claves encriptadas, que seria una funcionalidad necesaria por seguridad.
- Buscar algún tipo de marcador que indique que los targets masters están preparados para su despliegue, evitando posibles despliegues de masters aún no listos.
- Adaptar los scripts de gestión de targets al sistema de ficheros ZFS.

9. Glosario

ACL	Usado para especificar los derechos de acceso para los Initiators a los TPGs
BackStores	Proporciona el SCSI target con acceso generalizado a los dispositivos de almacenamiento de datos mediante su importación a través de los controladores de dispositivo correspondiente. No necesitan ser dispositivos SCSI físicos. Los tipos de medios backstore más importantes son bloques, ficheros y raw.
BTRFS	B-tree File System. Sistema de ficheros basado en el concepto Copy On Write(COW) y epleado en el mantenimiento de instantáneas(snapshots) del sistema operativo
CDB	Command Descriptor Block: El formato estandar de los comandos SCSI. Comunmente de longitud de 6,10 o 12 bytes, pueden ser de 16 bytes o de longitud variables.
CHAP	Command Descriptor Block: El formato estandar de los comandos SCSI. Comunmente de longitud de 6,10 o 12 bytes, pueden ser de 16 bytes o de longitud variables.
DAS	Direct-Attached Storage. Dispositivos de almacenamiento de datos conectados físicamente al equipo.
Demo-mode	Desactivación de autenticación para un iSCSI Endpoint. Deshabilita las ACLs, permite el acceso de solo-lectura a todos los iSCSI Initiator que intentan conectar a un Endpoint específico.
Endpoint	La combinación de un iSCSI TargetName con un iSCSI TPG Tag (IQN + Tag).
Fabric Modules	Los módulos fabric implementa el frontend de SCSI targe mediante la encapsulación y abstracción de las propiedades de las distintas interconexiones soportadas. Los siguientes módulos están disponibles: Fcoe, Fibre Channel, iSCSI, IEEE 1394, iSER, SRP y USB
FCP	Fibre Channel Protocol; protocolo de transport e usado para el envío y recepción de comandos SCSI en las redes FC
Fibre Chanel	Tecnología de red utilizada para redes de almacenamiento; proporciona conexiones de alta velocidad(de 2,4 hasta 16 GB/s) entre dispositivos de almacenamiento.
Hypervisor or VMM	Virtual Machine Monitor. Software, firmware o hardware encargado de gestionar, crear y ejecutar máquinas virtuales.
IPXE	Implementación de código abierto del firmware cliente Preboot eXecution Environment (PXE) y el bootloader, creada en 2010 como fork de gPXE. Se puede utilizar para permitir que equipos sin función de soporte PXE arranquen desde red o extiendan la implementación del cliente PXE

	soportando protocolos adicionales.
iSCSI	Capa de transporte definido en SCSI-3. Permite el uso de la interface SCSI sobre redes TCP/IP (RFC 3720)
KVM	Kernel-based Virtual Machine. Infraestructura de virtualización del núcleo Linux, lo que le otorga funciones de Hypervisor
Linux SCSI Target LIO	Implementación de código abierto de SCSI Target que se ha convertido en estándar una vez incluido en el Kernel de linux. Internamente, no inicia sesión, proporciona LUNs, y espera comandos SCSI desde un iniciator, realizando las transferencias de datos solicitadas. Implementa un SCSI target genérico que proporciona acceso remoto a la mayoría de tipos de almacenamiento de datos a través de los más comunes fabrics y protocolos.
LUN	Logical Unit Number utilizado para identificar una unidad lógica, dirigida a dispositivos con protocolo SCSI o SAN, como Fibre Channel o iSCSI
NAS	Network-Attached Storage. Red que proporciona acceso acceso al almacenamiento de datos a nivel de fichero. Utilizan protocolos para compartir ficheros en red como NFS, SMB o AFP.
PXE	PXE hace referencia al entorno de ejecución de prearranque (Preboot eXecution Environment). Es un entorno para arrancar e instalar el sistema operativo en ordenadores a través de una red, de manera independiente de los dispositivos de almacenamiento de datos disponibles (como discos duros) o de los sistemas operativos instalados.
SAN	Storage Area Network. Red que proporciona acceso al almacenamiento de datos a nivel de bloque.Los dispositivos de almacenamiento aparecen el el Sistema Operativo como dispositivos conectados localmente.
SCSI	Small Computer System Interface, (interfaz de sistema para pequeñas computadoras), Conjunto de estándares para conectar y transferir datos entre ordenadores y dispositivos periféricos físicamente. Definen comandos, protocolos, interfaces eléctricos y ópticos. (RFC 3783)
SCSI CDB	Command Descriptor Block. Comando SCSI enviado, contiene un byte con el código de operación seguido por cinco o más bytes con los parámetros del comando especificado. Una vez recibido y procesado el CDG el dispositivo retorna un byte con el status code y otra información.
SCSI Command	Arquitectura de comandos SCSI utiliza modelo de comunicación cliente-servidor. El PC es un cliente que realiza una petición al dispositivo de almacenamiento para realizar un servicio (lectura o escritura de datos).
SCSI Initiator	Endpoint que inicia una sesión SCSI que envía un comando SCSI
SCSI Target	Endpoint que no inicia una sesión SCSI, espera los comandos SCSI enviados por el Initiator y proporciona la transferencia de datos requerida. Proporciona los LUN a los inicializadores.

Snapshot	Copia del estado de un sistema operativo en un momento concreto.
targetcli	Es una interfaz de línea de comandos (CLI) para la de gestión de LIO. Es compatible con todos los módulos fabric y se basa en una arquitectura modular y extensible, con módulos plug-in para funcionalidad adicional. utiliza una biblioteca de target genérica subyacente a través de una API bien definida. Está implementado en Python.
Thin provisioning	Técnica mediante la cual, y haciendo uso de tecnología de virtualización, se consigue que una máquina aparente contar con más recursos físicos de los que dispone en realidad.

10. Bibliografia

IPXE – Open Source Boot Firmware:

<http://ipxe.org/start>

Wikipedia SCSI command:

https://en.wikipedia.org/wiki/SCSI_command

Linux Clustering.net:

<http://www.linuxclustering.net/2012/11/08/step-by-step-how-to-set-up-a-low-cost-san-with-the-linux-software-iscsi-target/>

Linuksovi blog:

<http://linuksovi.blogspot.com.es/2015/10/using-lioutils-to-create-storage-server.html>

Dedoimedo:

<http://www.dedoimedo.com/computers/btrfs-snapshots.html>

PXE Specification:

<http://download.intel.com/design/archives/wfm/downloads/pxespec.pdf>

Github targetmgr: Patrick Schmid:

<https://github.com/patschbo/targetmgr>

Github ipxe-phpmenu: skunkie.

<https://github.com/skunkie/ipxe-phpmenu>

11. Anexos

Anexo A. Comandos para el sistema de ficheros ZFS

- Crear un pool en /dev/sdb1

```
zpool create test -f /dev/sdb1
```

```
root@iscsiServer:~/mnt# zpool list
NAME SIZE ALLOC FREE EXPANDSZ FRAG CAP DEDUP HEALTH ALROOT
test 19,9G 64K 19,9G - 0% 0% 1.00x ONLINE -
root@iscsiServer:~/mnt# zpool status
pool: test
state: ONLINE
scan: none requested
config:

    NAME      STATE    READ WRITE CKSUM
    test       ONLINE   0     0     0
        sdb1   ONLINE   0     0     0

errors: No known data errors
```

El nuevo pool ZFS, de nombre *test* se crea en la raíz del SO si no se indica lo contrario en el comando.

- Crear un dataset en el pool /etc/test

```
zfs create test/data (donde data es el nombre del dataset)
```

```
root@iscsiServer:/test# zfs list
NAME      USED  AVAIL  REFER  MOUNTPOINT
test      81,5K 19,3G   19K    /test
test/data 19K   19,3G   19K    /test/data
```

- Creamos la estructura de datasets

Para el almacenamiento de los ficheros de respaldo de los Targets iSCSI:

```
/test
|__data/
|    |__master
|    |__images
```

```
zfs create test/data/master
zfs create test/data/images
```

```
root@iscsiServer:/test/data# zfs list
NAME      USED  AVAIL  REFER  MOUNTPOINT
test      126K 19,3G   19K    /test
test/data 57K   19,3G   19K    /test/data
test/data/images 19K 19,3G   19K    /test/data/images
test/data/master 19K 19,3G   19K    /test/data/master
```

```

root@iscsiServer:/test/data# ll
total 2
drwxr-xr-x 4 root root 4 may 22 12:26 ./
drwxr-xr-x 3 root root 3 may 22 12:12 ../
drwxr-xr-x 2 root root 2 may 22 12:26 images/
drwxr-xr-x 2 root root 2 may 22 12:25 master/

```

Copiamos uno de los masters del directorio `/opt/pool/master` en `/test/data/master`

```

root@iscsiServer:/test/data# cp /opt/pool/master/fedora.img /test/data/master/
root@iscsiServer:/test/data# ll master/
total 6948569
drwxr-xr-x 2 root root      3 may 22 12:31 ./
drwxr-xr-x 4 root root      4 may 22 12:26 ../
-rw----- 1 root root 16106127360 may 22 12:34 fedora.img

```

Ahora podría crearse un Target nuevo en Targetcli que apunta a `/test/data/master/fedora.img` que sería totalmente funcional.

- Clonación de un dataset

En BTRFS, realizábamos copias de un fichero maestro con la opción `-reflink="always"` del comando `cp` para poder desplegar tantos Targets como clientes tuviéramos.

Con el master en ZFS haremos un clon del dataset `/test/data/master`.

Primero, se crea una snapshot de dicho dataset, para después convertirlo en un clon. El snapshot es un fichero de sólo lectura, por lo que no es útil a la hora de exportarlo como Target, para eso necesitaremos convertirlo en un clon.

Snapshot: `zfs snapshot test/data/master@fedoraMaster`

```

root@iscsiServer:/test/data# zfs list -t snapshot
NAME                               USED  AVAIL  REFER  MOUNTPOINT
test/data/master@fedoraMaster      0    - 6,63G  -

```

Clon: `zfs clone /test/data/master@fedoraMaster /test/data/images/fedoraA`

Se clona el snapshot creado en el paso anterior y se crea un nuevo dataset en `/test/data/images`, de nombre `fedoraA` que contiene el fichero `fedora.img`

```

root@iscsiServer:/test/data# ll images/
total 2
drwxr-xr-x 3 root root 3 may 22 13:08 ./
drwxr-xr-x 4 root root 4 may 22 12:26 ../
drwxr-xr-x 2 root root 3 may 22 12:31 fedoraA/
root@iscsiServer:/test/data# ll images/fedoraA/
total 6948571
drwxr-xr-x 2 root root      3 may 22 12:31 ./
drwxr-xr-x 3 root root      3 may 22 13:08 ../
-rw----- 1 root root 16106127360 may 22 12:45 fedora.img

```

Creamos un nuevo Target, respaldado por `/test/data/images/fedoraA/fedora.img`

```
0- backstores ..... [....]
  0- fileio ..... [5 Storage Objects]
    | 0- disk ..... [15.0G, /opt/pool/master/disk.img, in use]
    | 0- fedora ..... [15.0G, /opt/pool/master/fedora.img, in use]
    | 0- fedoraB ..... [15.0G, /opt/pool/images/fedoraB.img, in use]
    | 0- fedoraz ..... [15.0G, /test/data/master/fedora.img, in use]
    | 0- fedorazB ..... [15.0G, /test/data/images/fedoraA/fedora.img, in use]
  0- iblock ..... [0 Storage Object]
  0- pscsi ..... [0 Storage Object]
  0- rd_mcp ..... [0 Storage Object]
0- iqn.2016-05.org.example:fedorazB ..... [1 TPG]
  0- tpg1 ..... [enabled]
    0- acls ..... [0 ACLs]
    0- luns ..... [1 LUN]
      | 0- lun0 ..... [fileio/fedorazB (/test/data/images/fedoraA/fedora.img)]
    0- portals ..... [1 Portal]
      0- 10.1.0.1:3260 ..... [OK, iser disabled]
```

-Escritura de un fichero de 4GB en un Target:

Target guardado en un sistema de ficheros ZFS:

```
[user@localhost ~]$ time dd if=/dev/zero of=/home/user/fichero bs=1M count=4096
4096+0 registros leídos
4096+0 registros escritos
4294967296 bytes (4,3 GB) copiados, 124,63 s, 34,5 MB/s

real    2m4.655s
user    0m0.019s
sys     0m8.365s
```

Target guardado en un sistema de ficheros BTRFS:

```
[user@localhost ~]$ time dd if=/dev/zero of=/home/user/fichero bs=1M count=4096
4096+0 registros leídos
4096+0 registros escritos
4294967296 bytes (4,3 GB) copiados, 74,2935 s, 57,8 MB/s

real    1m14.505s
user    0m0.021s
sys     0m8.814s
```

Por defecto, todos los Targets creados en Targetcli usan el método *non-buffered* para la escritura de datos sobre el FileIO, es decir, cualquier dato que se escribe en el disco duro sustentado por dicho disco duro se almacena directamente, sin pasar por un búffer. Los cambios en el disco se producen “en tiempo real”, por lo que una desconexión con el servidor iSCSI no debería provocar pérdida de datos; por contra, si se activa el modo *buffered*, los datos se almacena en caché y quedan a la espera de ser transferidos al disco duro para su almacenamiento. Una pérdida de conexión con el servidor iSCSI podría suponer pérdida de datos, pero este modo proporciona un incremento en el rendimiento (según la documentación oficial del servicio)

Creamos un Target con la opción *buffered* activada:

```
/backstores/fileio> create fedora_buffer /opt/pool/images/fedora_buffer.img 15G buffered=true
Using buffered mode.
Created fileio fedora_buffer.
```

Target con buffered=true almacenado en un sistema de ficheros BTRFS:

```
[User@localhost Escritorio]$ time dd if=/dev/zero of=/home/user/fichero bs=1M count=4096
4096+0 registros leidos
4096+0 registros escritos
4294967296 bytes (4,3 GB) copiados, 74,9121 s, 57,3 MB/s

real    1m14.914s
user    0m0.016s
sys     0m9.455s
```

Target con buffered=true almacenado en un sistema de ficheros ZFS:

Primero clonamos el snapshot creado previamente con el comando

```
zfs clone test/data/master@fedoraMaster test/data/images/fedoraBuffer
```

Realizamos la misma prueba de escritura para este Target:

```
[user@localhost ~]$ time dd if=/dev/zero of=/home/user/fichero bs=1M count=4096
4096+0 registros leidos
4096+0 registros escritos
4294967296 bytes (4,3 GB) copiados, 72,2504 s, 59,4 MB/s

real    1m12.257s
user    0m0.026s
sys     0m8.968s
```


Anexo B. Scripts Python

tgtcreatemaster

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
"""
```

```
ISCSI Target Manager - tgtcreatemaster
```

```
tgtcreatemaster. Utilizar para crear masters y desplegar en iscsi
```

```
Opciones:
```

```
    -m master      Nombre del master a crear.
                   - No incluir puntos (.) ni subrayados (_)

    -s size        Tamaño del fileio del master.

    -h             Muestra la ayuda
```

```
Ejemplo:
```

```
    tgtcreatemaster -m UbuntuServer16-04 -s 10G
```

```
"""
```

```
import logging
import ConfigParser
import getopt
import sys
import os
import rtslib

# ---- Recuperar datos del fichero de configuracion ----
config = ConfigParser.ConfigParser()
config.read('/etc/targetmgr/targetmgr.conf')

SERVER = config.get('SERVER', 'server')
PORT = config.get('SERVER', 'port')

PATH_LOG = config.get('PATH', 'path_log')
PATH_MASTER = config.get('PATH', 'path_master')
FILE_LOG = PATH_LOG + 'targetmgr.log'

PWD_MASTER = config.get('MASTER', 'pwd_master')

IQN = config.get('IQN', 'iqn_prefix',) + "." + config.get('MASTER', 'iqn_naming_master')
+ ":"

# ---- Crear logger ----
FORMAT_DATA='%Y-%d-%m %H:%M:%S'
FORMAT='% (asctime)s %(message)s'
logging.basicConfig(filename=FILE_LOG,format=FORMAT,level=logging.INFO,datefmt=FORMAT_DATA)

# ----- Funciones control de parametros -----
def error_msg(str_error):
    """
    Mostrar mensaje de error antes de salir
    """
    print "\nERROR: " + str_error + "\n"
    sys.exit(1)

def check_master_exist(master):
    """
    Verificar si el fichero master existe
    """
    return os.path.isfile(PATH_MASTER + master + '.img')

def check_iqn(iqn):
```

```

"""
Verificar si la cadena iqn es correcta
"""
return rtplib.utils.is_valid_wwn('iqn',iqn)

def check_size_incorrect(size):
"""
Verificar parametro correcto size
"""
return False

# ----- iSCSI Funciones crear fileio-----

def _next_free_backstore_index():
    backstore_indices = [backstore.index for backstore in
rtplib.root.RTSRoot().backstores]
    if not backstore_indices:
        next_free_index = 0
    else:
        next_free_index = max(backstore_indices) + 1
    return next_free_index

def create_fileio(iqn, master, size):
    if '_' + master in (lun['name'] for lun in current_fileios()):
        error_msg("nombre backstores/fileio '%s' ya existe." % master)
    elif master + '.img' in (lun['path'] for lun in current_fileios()):
        error_msg("imagen fichero master: %s ya existe. " % master + '.img')
    else:
        backstore = rtplib.FileIOBackstore(_next_free_backstore_index(), mode='create')
        try:
            rtplib.FileIOStorageObject(backstore, '_' + master, PATH_MASTER + master +
'.img', size)
        except:
            error_msg('No es posible crear backstore fileio %s' % iqn)
            backstore.delete()
            raise

    return None

def current_fileios():
    existing_fileios = []
    for backstore in rtplib.root.RTSRoot().backstores:
        for storage_object in backstore.storage_objects:
            existing_fileios.append(
                {
                    'name': storage_object.name,
                    'size': storage_object.size,
                    'path': storage_object.udev_path,
                    'index': storage_object.backstore.index,
                }
            )

    return existing_fileios

# ---- iSCSI Funciones crear target iscsi -----

def current_targets():
    existing_targets = []
    for target in rtplib.FabricModule('iscsi').targets:
        existing_targets.append(target.wwn,)
    return existing_targets

def create_target(iqn):
    if iqn in current_targets():
        error_msg('iscsi-target "%s" ya existe.' % iqn)
    else:
        try:
            rtplib.Target(rtplib.FabricModule('iscsi'), wwn=iqn)
        except:
            raise

    return None

# ----- iSCSI Funciones crear tpg -----

```

```

def create_tpg(iqn):
    try:
        rtplib.TPG(rtplib.Target(rtplib.FabricModule('iscsi'),iqn), 1)._set_enable(True)
    except:
        raise

    return None

def _get_single_tpg(iqn):
    tpg = rtplib.TPG(rtplib.Target(rtplib.FabricModule('iscsi'), iqn), 1)

    return tpg

def set_custom_tpg_attributes(iqn):
    # attributes
    tpg_attributes = dict(
        authentication = '0',
        demo_mode_write_protect='0',
        cache_dynamic_acls='1',
        generate_node_acls='1',
    )
    for attribute, value in tpg_attributes.items():
        if _get_single_tpg(iqn).get_attribute(attribute) != value:
            _get_single_tpg(iqn).set_attribute(attribute, value)

    # auth attributes
    _get_single_tpg(iqn).set_auth_attr('userid','master')
    _get_single_tpg(iqn).set_auth_attr('password',PWD_MASTER)

def create_portal(iqn):
    _get_single_tpg(iqn).network_portal(SERVER, PORT, mode='any')

# ----- ISCSI funciones crear Luns -----

def _next_free_lun_index(iqn):
    lun_indices = [lun.lun for lun in _get_single_tpg(iqn).luns]
    if not lun_indices:
        next_free_index = 0
    else:
        next_free_index = max(lun_indices) + 1

    return next_free_index

def current_attached_luns(iqn):
    attached_luns = []
    for lun in _get_single_tpg(iqn).luns:
        attached_luns.append(
            {
                'lun': lun.lun,
                'name': lun.storage_object.name,
                'device': lun.storage_object.udev_path,
            }
        )

    return attached_luns

def _get_storage_object_fileio(iqn):
    for backstore in rtplib.root.RTSRoot().backstores:
        for storage_object in backstore.storage_objects:
            if storage_object.name == iqn:
                return storage_object

    return False

def create_attached_lun(iqn, master):
    if iqn in (curlun['name'] for curlun in current_attached_luns(iqn)):
        error_msg("LUN %s ya enlazado" % iqn)
    else:
        try:
            rtplib.LUN(
                _get_single_tpg(iqn),_next_free_lun_index(iqn),
                storage_object=_get_storage_object_fileio('_' + master)
            )
        except:

```

```

        error_msg('No es posible al enlazar el fileio al LUN en %s' % iqn)
        raise

    return None

# ----- Guardar configuración en targetcli -----

def save_to_disk():
    os.system("echo -e '\nsaveconfig\nY' | targetcli")

# ----- Main Function -----

if __name__ == '__main__':

    try:
        opts, args = getopt.getopt(sys.argv[1:], "hm:s:", ["help"])
    except getopt.GetoptError:
        error_msg("Parametros incorrectos")

    # Iniciar variables de los argumentos
    master = ''
    size = ''

    # Recuperar variables de la linea de comandos
    for opt, arg in opts:
        if opt in ("-h", "--help"):
            print
            print __doc__
            sys.exit()
        elif opt in ('-m'):
            master = arg
        elif opt in ('-s'):
            size = arg

    # Chequear parametros obligatorios
    if not master:
        error_msg("Parametro -m obligatorio. Ejemplo -m Ubuntu_Server_14-4")
    if not size:
        error_msg("Parametro -s obligatorio. Ejemplo -s 10G")

    # Verificar parametro -m

    if '.' in master:
        error_msg("Nombre del master incorrecto")

    # Verificar parametro -s
    if check_size_incorrect(size):
        error_msg("Valor incorrecto en el parámetro Size")

    # Verificar iqn generado
    iqn = IQN + master
    if not rtplib.utils.is_valid_wwn('iqn',iqn):
        error_msg("%s no es un identificador wwn valido" % iqn)

    # Iniciar despliegue

    create_target(iqn)
    create_tpg(iqn)
    set_custom_tpg_attributes(iqn)
    create_portal(iqn)
    create_fileio(iqn, master, size)
    create_attached_lun(iqn, master)
    logging.info("[CREATEMASTER] iscsi: " + iqn + " fileio: " + "_" + master + " file: "
+ master + ".img")

    save_to_disk()
    print "\n[ OK ] Master creado correctamente ver fichero log para ver detalles.\n"

```

tgtclonemaster

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
"""
```

ISCSI Target Manager - tgtclonemaster

tgtclonemaster. Utilizado para clonar un master a otro master

Opciones:

```
-m master      Nombre del master a clonar
-c clone       Nombre del clone
                - No incluir puntos ni subrayados

-h            Muestra la ayuda
```

Ejemplo:

```
tgtclonemaster -m master -c clone
```

```
"""
```

```
import logging
import ConfigParser
import getopt
import sys
import os
import rtslib
```

```
# ---- Recuperar datos del fichero de configuracion ----
```

```
config = ConfigParser.ConfigParser()
config.read('/etc/targetmgr/targetmgr.conf')
```

```
SERVER = config.get('SERVER', 'server')
PORT = config.get('SERVER', 'port')
```

```
PATH_LOG = config.get('PATH', 'path_log')
PATH_MASTER = config.get('PATH', 'path_master')
FILE_LOG = PATH_LOG + 'targetmgr.log'
```

```
PWD_MASTER = config.get('MASTER', 'pwd_master')
```

```
IQN = config.get('IQN', 'iqn_prefix',) + "." + config.get('MASTER', 'iqn_naming_master')
+ ":"
```

```
# ---- Crear logger ----
```

```
FORMAT_DATA='%Y-%d-%m %H:%M:%S'
FORMAT='%(%asctime)s %(message)s'
logging.basicConfig(filename=FILE_LOG,format=FORMAT,level=logging.INFO,datefmt=FORMAT_DATA)
```

```
# ---- Funciones control de parametros ----
```

```
def error_msg(str_error):
```

```
    """
    Mostrar mensaje de error antes de salir
    """
    print "\nERROR: " + str_error + "\n"
    sys.exit(1)
```

```
def check_file_exist(master):
```

```
    """
    Verificar si el fichero master existe
    """
    return os.path.isfile(PATH_MASTER + master + '.img')
```

```
def check_iqn(iqn):
```

```
    """
    Verificar si la cadena iqn es correcta
    """
    return rtslib.utils.is_valid_wnn('iqn',iqn)
```

```

def check_size_incorrect(size):
    """
    Verificar parametro correcto size
    """
    return False

# ----- iSCSI Funciones crear fileio-----

def _next_free_backstore_index():
    backstore_indices = [backstore.index for backstore in
rtslib.root.RTSRoot().backstores]
    if not backstore_indices:
        next_free_index = 0
    else:
        next_free_index = max(backstore_indices) + 1
    return next_free_index

def create_fileio(iqn, clone, size):
    if '_' + clone in (lun['name'] for lun in current_fileios()):
        error_msg("nombre backstores/fileio '%s' ya existe." % clone)
    elif clone + '.img' in (lun['path'] for lun in current_fileios()):
        error_msg("imagen fichero clone: %s ya existe. " % clone + '.img')
    else:
        backstore = rtslib.FileIOBackstore(_next_free_backstore_index(), mode='create')
        try:
            rtslib.FileIOStorageObject(backstore, '_' + clone, PATH_MASTER + clone +
'.img', size)
        except:
            error_msg('No es posible crear backstore fileio %s' % iqn)
            backstore.delete()
            raise

    return None

def current_fileios():
    existing_fileios = []
    for backstore in rtslib.root.RTSRoot().backstores:
        for storage_object in backstore.storage_objects:
            existing_fileios.append(
                {
                    'name': storage_object.name,
                    'size': storage_object.size,
                    'path': storage_object.udev_path,
                    'index': storage_object.backstore.index,
                }
            )

    return existing_fileios

# ---- iSCSI Funciones crear target iscsi -----

def current_targets():
    existing_targets = []
    for target in rtslib.FabricModule('iscsi').targets:
        existing_targets.append(target.wwn,)
    return existing_targets

def create_target(iqn):
    if iqn in current_targets():
        error_msg('iscsi-target "%s" ya existe.' % iqn)
    else:
        try:
            rtslib.Target(rtslib.FabricModule('iscsi'), wwn=iqn)
        except:
            raise

    return None

# ----- iSCSI Funciones crear tpg -----

def create_tpg(iqn):
    try:
        rtslib.TPG(rtslib.Target(rtslib.FabricModule('iscsi'),iqn), 1)._set_enable(True)
    except:

```

```

        raise

    return None

def _get_single_tpg(iqn):
    tpg = rtslib.TPG(rtslib.Target(rtslib.FabricModule('iscsi'), iqn), 1)

    return tpg

def set_custom_tpg_attributes(iqn):
    # attributes
    tpg_attributes = dict(
        authentication = '0',
        demo_mode_write_protect='0',
        cache_dynamic_acls='1',
        generate_node_acls='1',
    )
    for attribute, value in tpg_attributes.items():
        if _get_single_tpg(iqn).get_attribute(attribute) != value:
            _get_single_tpg(iqn).set_attribute(attribute, value)

    # auth attributes
    _get_single_tpg(iqn).set_auth_attr('userid', 'master')
    _get_single_tpg(iqn).set_auth_attr('password', PWD_MASTER)

def create_portal(iqn):
    _get_single_tpg(iqn).network_portal(SERVER, PORT, mode='any')

# ----- ISCSI funciones crear Luns -----

def _next_free_lun_index(iqn):
    lun_indices = [lun.lun for lun in _get_single_tpg(iqn).luns]
    if not lun_indices:
        next_free_index = 0
    else:
        next_free_index = max(lun_indices) + 1

    return next_free_index

def current_attached_luns(iqn):
    attached_luns = []
    for lun in _get_single_tpg(iqn).luns:
        attached_luns.append(
            {
                'lun': lun.lun,
                'name': lun.storage_object.name,
                'device': lun.storage_object.udev_path,
            }
        )

    return attached_luns

def _get_storage_object_fileio(iqn):
    for backstore in rtslib.root.RTSRoot().backstores:
        for storage_object in backstore.storage_objects:
            if storage_object.name == iqn:
                return storage_object

    return False

def create_attached_lun(iqn, clone):
    if iqn in (curlun['name'] for curlun in current_attached_luns(iqn)):
        error_msg("LUN %s ya enlazado" % iqn)
    else:
        try:
            rtslib.LUN(
                _get_single_tpg(iqn), _next_free_lun_index(iqn),
                storage_object=_get_storage_object_fileio('_' + clone)
            )
        except:
            error_msg('No es posible al enlazar el fileio al LUN en %s' % iqn)
            raise

    return None

```

```

# ----- Clonar la imagen -----

def copy_image(master, clone):
    file_from = PATH_MASTER + master + '.img'
    file_to = PATH_MASTER + clone + '.img'

    cmd = 'cp --reflink="always" ' + file_from + ' ' + file_to
    os.system(cmd)

# ----- Guardar configuración en targetcli -----

def save_to_disk():
    os.system("echo -e '\nsaveconfig\nY' | targetcli")

# ----- Main Function -----

if __name__ == '__main__':

    try:
        opts, args = getopt.getopt(sys.argv[1:], "hm:c:", ["help"])
    except getopt.GetoptError:
        error_msg("Parametros incorrectos")

    # Iniciar variables de los argumentos
    master = ''
    clone = ''

    # Recuperar variables de la linea de comandos
    for opt, arg in opts:
        if opt in ("-h", "--help"):
            print
            print __doc__
            sys.exit()
        elif opt in ('-m'):
            master = arg
        elif opt in ('-c'):
            clone = arg

    # Chequear parametros obligatorios
    if not master:
        error_msg("Parametro -m master obligatorio. ")
    if not clone:
        error_msg("Parametro -c clone obligatorio. ")

    # Verificar parametro -m
    if not check_file_exist(master):
        error_msg("fichero Master no existente")

    # Verificar parametro -s
    if check_file_exist(clone):
        error_msg("fichero Clone ya existente")

    if '.' in clone:
        error_msg("nombre del Clone incorrecto")

    # Verificar iqn generado
    iqn = IQN + clone
    if not rtlib.utils.is_valid_wwn('iqn',iqn):
        error_msg("%s no es un identificador wwn valido" % iqn)

    # Iniciar despliegue
    size_master= os.path.getsize(PATH_MASTER + master + '.img')

    create_target(iqn)
    create_tpg(iqn)
    set_custom_tpg_attributes(iqn)
    create_portal(iqn)
    create_fileio(iqn,clone,size_master)
    create_attached_lun(iqn,clone)

    copy_image(master,clone)

    logging.info("[CLONEMASTER] iscsi: " + iqn + " fileio: " + "_" + clone + " file: " +
clone + ".img From master: " + master + ".img" )

```



```
save_to_disk()
print "\n[ OK ] Master clonado correctamente ver fichero log para ver detalles.\n"
```

tgtcloneimage

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
```

```
"""
```

ISCSI Target Manager - tgtcloneimage

tgtcloneimage. Utilizar para clonar un master o imagen y desplegar el target.

Opciones:

```
[- m master|-i image] [-u user|-f file_users] [-n naming-authority]
```

-m master Nombre del master a clonar.

-i image naming-authority y unique name de la imagen a clonar

-u usuario Usuario asignado al target a desplegar

-f file_users Fichero de usuario para despliegue por lotes

-n naming-auth Naming-authority del target a desplegar

-h Muestra la ayuda

Ejemplos:

```
tgtcloneimage -m master -u usuario -n xx.yy.zz
tgtcloneimage -i xx.yy.zz:usuario -f usuarios -n xx.yy.zz
```

```
"""
```

```
import logging
import ConfigParser
import getopt
import sys
import os
import rtslib
```

```
# ---- Recuperar datos del fichero de configuracion ----
FILE_USERS = '/etc/targetmgr/users'
```

```
config = ConfigParser.ConfigParser()
config.read('/etc/targetmgr/targetmgr.conf')
```

```
SERVER = config.get('SERVER', 'server')
PORT = config.get('SERVER', 'port')
```

```
PATH_LOG = config.get('PATH', 'path_log')
PATH_MASTER = config.get('PATH', 'path_master')
PATH_IMAGES = config.get('PATH', 'path_images')
FILE_LOG = PATH_LOG + 'targetmgr.log'
```

```
IQN = config.get('IQN', 'iqn_prefix',) + '.'
```

```
# ---- Crear logger ----
```

```
FORMAT_DATA='%Y-%d-%m %H:%M:%S'
FORMAT='% (asctime)s %(message)s'
logging.basicConfig(filename=FILE_LOG,format=FORMAT,level=logging.INFO,datefmt=FORMAT_DATA)
```

```
# ---- Funciones control de parametros ----
```

```
def error_msg(str_error):
    print "\nERROR: " + str_error + "\n"
    sys.exit(1)
```

```

def check_master_exist(master):
    return os.path.isfile(PATH_MASTER + master + '.img')

def _naming_to_path(str_n):
    return PATH_IMAGES + (str_n.replace(':', '_')).replace('.', '/') + '.img'

def check_image_exist(image):
    return os.path.isfile(scan_image_path(image))

def check_iqn(iqn):
    return rtplib.utils.is_valid_wwn('iqn', iqn)

def check_file_users_exist(file):
    return os.path.isfile(file)

def check_user(user):
    f_users = open(FILE_USERS, 'r')

    check=False
    while True:
        linea = f_users.readline().rstrip('\n')
        if not linea:
            break

        str_user = linea.split(':')
        if str_user[0] == user:
            check = True
            break

    f_users.close()
    return check

def check_users(file_user):
    f_users = open(file_user, 'r')
    while True:
        user = f_users.readline().rstrip('\n')
        if not user:
            break

        if not check_user(user):
            f_users.close()
            return False

    f_users.close()
    return True

# ----- Funciones de rutas y ficheros -----

def get_password(user):
    f_users = open(FILE_USERS, 'r')
    while True:
        linea = f_users.readline().rstrip('\n')
        if not linea:
            break

        str_user = linea.split(':')
        if str_user[0] == user:
            f_users.close()
            return str_user[1]

    f_users_close()

    return None

def scan_path_file(iqn):
    split_iqn = iqn.split('.')
    file_path = PATH_IMAGES
    for i in range(2, len(split_iqn)-1):
        file_path += split_iqn[i] + '/'
    file_name = split_iqn[len(split_iqn)-1].replace(':', '_') + '.img'

    return {'path':file_path, 'file':file_name}

```

```

def scan_path(iqn):
    path_iqn=scan_path_file(iqn)
    return path_iqn['path'] + path_iqn['file']

def scan_image_path(image):
    split_image = image.split('.')
    file_path = PATH_IMAGES
    for i in range(0,len(split_image)-1):
        file_path += split_image[i] + '/'
    file_name = split_image[len(split_image)-1].replace(':', '_') + '.img'
    return file_path + file_name

def scan_image_master(image):
    split_image = image.split('.')
    return split_image[len(split_image)-1].split(':')[0]

# ----- iSCSI Funciones crear fileio-----

def _next_free_backstore_index():
    backstore_indices = [backstore.index for backstore in
    rtplib.root.RTSRoot().backstores]
    if not backstore_indices:
        next_free_index = 0
    else:
        next_free_index = max(backstore_indices) + 1
    return next_free_index

def create_fileio(iqn, name, size):
    if name in (lun['name'] for lun in current_fileios()):
        error_msg("nombre backstores/fileio '%s' ya existe." % master)
    elif scan_path(iqn) in (lun['path'] for lun in current_fileios()):
        error_msg("fichero imagen a clonar: %s ya existe. " % scan_path(iqn))
    else:
        backstore = rtplib.FileIOBackstore(_next_free_backstore_index(), mode='create')
        try:
            if not os.path.isdir(scan_path_file(iqn)['path']):
                os.makedirs(scan_path_file(iqn)['path'])
            rtplib.FileIOStorageObject(backstore,name, scan_path(iqn), size)
        except:
            error_msg('No es posible crear backstore fileio %s' % iqn)
            backstore.delete()
            raise

    return None

def current_fileios():
    existing_fileios = []
    for backstore in rtplib.root.RTSRoot().backstores:
        for storage_object in backstore.storage_objects:
            existing_fileios.append(
                {
                    'name': storage_object.name,
                    'size': storage_object.size,
                    'path': storage_object.udev_path,
                    'index': storage_object.backstore.index,
                }
            )

    return existing_fileios

# ---- iSCSI Funciones crear target iscsi -----

def current_targets():
    existing_targets = []
    for target in rtplib.FabricModule('iscsi').targets:
        existing_targets.append(target.wwn,)
    return existing_targets

def create_target(iqn):
    if iqn in current_targets():
        error_msg('iscsi-target "%s" ya existe.' % iqn)
    else:
        try:

```

```

        rtplib.Target(rtplib.FabricModule('iscsi'), wwn=iqn)
    except:
        raise

    return None

# ----- iSCSI Funciones crear tpg -----

def create_tpg(iqn):
    try:
        rtplib.TPG(rtplib.Target(rtplib.FabricModule('iscsi'),iqn), 1)._set_enable(True)
    except:
        raise

    return None

def _get_single_tpg(iqn):
    tpg = rtplib.TPG(rtplib.Target(rtplib.FabricModule('iscsi'), iqn), 1)

    return tpg

def set_custom_tpg_attributes(iqn,user):
    # attributes
    tpg_attributes = dict(
        authentication = '0',
        demo_mode_write_protect='0',
        cache_dynamic_acls='1',
        generate_node_acls='1',
    )
    for attribute, value in tpg_attributes.items():
        if _get_single_tpg(iqn).get_attribute(attribute) != value:
            _get_single_tpg(iqn).set_attribute(attribute, value)

    # auth attributes
    _get_single_tpg(iqn).set_auth_attr('userid',user)
    _get_single_tpg(iqn).set_auth_attr('password',get_password(user))

def create_portal(iqn):
    _get_single_tpg(iqn).network_portal(SERVER, PORT, mode='any')

# ----- iSCSI funciones crear Luns -----

def _next_free_lun_index(iqn):
    lun_indices = [lun.lun for lun in _get_single_tpg(iqn).luns]
    if not lun_indices:
        next_free_index = 0
    else:
        next_free_index = max(lun_indices) + 1

    return next_free_index

def current_attached_luns(iqn):
    attached_luns = []
    for lun in _get_single_tpg(iqn).luns:
        attached_luns.append(
            {
                'lun':    lun.lun,
                'name':   lun.storage_object.name,
                'device': lun.storage_object.udev_path,
            }
        )

    return attached_luns

def _get_storage_object_fileio(iqn):
    for backstore in rtplib.root.RTSRoot().backstores:
        for storage_object in backstore.storage_objects:
            if storage_object.name == iqn:
                return storage_object

    return False

def create_attached_lun(iqn, name):
    if name in (curlun['name'] for curlun in current_attached_luns(iqn)):

```

```

        error_msg("LUN %s ya enlazado" % iqn)
    else:
        try:
            rtplib.LUN(
                _get_single_tpg(iqn), _next_free_lun_index(iqn),
                storage_object=_get_storage_object_fileio(name)
            )
        except:
            error_msg('No es posible al enlazar el fileio al LUN en %s' % iqn)
            raise

    return None

# ----- Clonar la imagen -----

def copy_image(file_from, file_to):
    cmd = 'cp --reflink="always" ' + file_from + ' ' + file_to
    os.system(cmd)

# ----- Guardar configuración en targetcli -----

def save_to_disk():
    os.system("echo -e '\nsaveconfig\nY' | targetcli")

# ----- Main Function -----

if __name__ == '__main__':
    try:
        opts, args = getopt.getopt(sys.argv[1:], "hm:i:u:f:n:", ["help"])
    except getopt.GetoptError:
        error_msg("Parametros incorrectos")

    # Iniciar variables de los argumentos
    user = ''
    master = ''
    image = ''
    file_users = ''
    naming_auth = ''

    # Recuperar variables de la linea de comandos
    for opt, arg in opts:
        if opt in ("-h", "--help"):
            print
            print __doc__
            sys.exit()
        elif opt in ('-m'):
            master = arg
        elif opt in ('-i'):
            image = arg
        elif opt in ('-u'):
            user = arg
        elif opt in ('-f'):
            file_users = arg
        elif opt in ('-n'):
            naming_auth = arg

    # Chequear parametros -m master y -i imagen
    if master and image:
        error_msg("Elegir copia de master (-m) o imagen (-i).")
    elif image and not master:
        if not check_image_exist(image):
            error_msg("La imagen indicada no existe.")
    elif not image and master:
        if not check_master_exist(master):
            error_msg("El master indicado no existe.")
    else:
        error_msg("Parámetros [-m / -i ] obligatorios.")

    # Chequear parametros -u user y -f file_users
    if user and file_users:
        error_msg("Elegir despliegue individual (-u) o desde fichero (-f).")
    elif user and not file_users:

```

```

        if not check_user(user):
            error_msg("El usuario no existe como usuario de targetmgr.")
    elif not user and file_users:
        if not check_file_users_exist(file_users):
            error_msg("El fichero de usuarios indicado no existe.")
        if not check_users(file_users):
            error_msg("Usuario no encontrado como usuario de targetmgr.")
    else:
        error_msg("Parámetros [ -u / -f ] obligatorios.")

# Chequear parametros obligatorios
if not naming_auth:
    error_msg("Parametro -n obligatorio. Ejemplo -n xx.yy.zz")

if user:
    # Verificar iqn generado
    if master:
        iqn = IQN + naming_auth + '.' + master + ':' + user
    elif image:
        iqn = IQN + naming_auth + '.' + scan_image_master(image) + ':' + user

    if not check_iqn(iqn):
        error_msg("%s no es un identificador wwn valido" % iqn)

    # Iniciar despliegue usuario

    create_target(iqn)
    create_tpg(iqn)
    set_custom_tpg_attributes(iqn, user)
    create_portal(iqn)

    if master:
        from_file = PATH_MASTER + master + '.img'
        size_fileio = os.path.getsize(from_file)
        name_fileio = naming_auth + '.' + master + ':' + user
    elif image:
        from_file = scan_image_path(image)
        size_fileio = os.path.getsize(from_file)
        name_fileio = naming_auth + '.' + scan_image_master(image) + ':' + user

    create_fileio(iqn, name_fileio, size_fileio)
    create_attached_lun(iqn, name_fileio)

    copy_image(from_file, scan_path(iqn))

    save_to_disk()
    logging.info("[CLONEIMAGE] iscsi:" + iqn + " fileio: " + name_fileio + " from: " +
+ from_file + " to: " + scan_path(iqn))
    print "\n[ OK ] Target añadido correctamente ver fichero log para ver
detalles.\n"
    sys.exit(0)

if file_users:
    f_users = open(file_users, 'r')
    while True:
        user = f_users.readline().rstrip('\n')
        if not user:
            break

    # Verificar iqn generado
    if master:
        iqn = IQN + naming_auth + '.' + master + ':' + user
    elif image:
        iqn = IQN + naming_auth + '.' + scan_image_master(image) + ':' + user

    if not check_iqn(iqn):
        error_msg("%s no es un identificador wwn valido" % iqn)

    # Iniciar despliegue usuario

    create_target(iqn)
    create_tpg(iqn)
    set_custom_tpg_attributes(iqn, user)
    create_portal(iqn)

```

```

if master:
    from_file = PATH_MASTER + master + '.img'
    size_fileio = os.path.getsize(from_file)
    name_fileio = naming_auth + '.' + master + ':' + user
elif image:
    from_file = scan_image_path(image)
    size_fileio = os.path.getsize(from_file)
    name_fileio = naming_auth + '.' + scan_image_master(image) + ':' + user

create_fileio(iqn, name_fileio, size_fileio)
create_attached_lun(iqn, name_fileio)

copy_image(from_file, scan_path(iqn))

logging.info("[CLONEIMAGE+] iscsi:" + iqn + " fileio: " + name_fileio + "
from: " + from_file + " to: " + scan_path(iqn))

save_to_disk()
print "\n[ OK ] Targets añadidos correctamente ver fichero log para ver
detalles.\n"
f_users.close()

sys.exit(0)

```

tgtdeletemaster

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-

```

"""

ISCSI Target Manager - tgtdeletemaster

**tgtdeletemaster. Utilizar para replegar target iSCSI masters y
eliminar su imagen asociada del sistema de archivos**

Opciones:

```

-m master      Nombre de la imagen master a replegar
                - No incluir extensión .img

-b            Crear un backup de la imagen master.

-h            Muestra la ayuda

```

Ejemplo:

```

tgtdeletemaster -m master -b

```

"""

```

import logging
import ConfigParser
import getopt
import sys
import os
import rtslib
import datetime

```

---- Recuperar datos del fichero de configuracion ----

```

FILE_TARGETMGR= '/etc/targetmgr/targetmgr.conf'

```

```

config = ConfigParser.ConfigParser()
config.read(FILE_TARGETMGR)

```

```

SERVER = config.get('SERVER', 'server')
PORT = config.get('SERVER', 'port')

```

```

PATH_LOG = config.get('PATH', 'path_log')
PATH_MASTER = config.get('PATH', 'path_master')
FILE_LOG = PATH_LOG + 'targetmgr.log'

```

```

PWD_MASTER = config.get('MASTER', 'pwd_master')

```

```

IQN = config.get('IQN', 'iqn_prefix',) + "." + config.get('MASTER', 'iqn_naming_master')
+ ":"

# ---- Crear logger ----

FORMAT_DATA='%Y-%d-%m %H:%M:%S'
FORMAT='% (asctime)s %(message)s'
logging.basicConfig(filename=FILE_LOG,format=FORMAT,level=logging.INFO,datefmt=FORMAT_DATA)

# ----- Funciones control de parametros -----

def error_msg(str_error):
    print "\nERROR: " + str_error + "\n"
    sys.exit(1)

def check_master_exist(master):
    return os.path.isfile(PATH_MASTER + master + '.img')

def check_iqn(iqn):
    return rtplib.utils.is_valid_wwn('iqn',iqn)

# ----- iSCSI Funciones borrar fileio y target -----

def current_targets():
    existing_targets = []
    for target in rtplib.FabricModule('iscsi').targets:
        existing_targets.append(target.wwn,)
    return existing_targets

def delete_fileio(name, backup):
    for backstore in rtplib.root.RTSRoot().backstores:
        for storage_object in backstore.storage_objects:
            if storage_object.name == name:
                if backup:
                    file_backup = storage_object.udev_path.replace('.img', '.' +
datetime.datetime.now().strftime('%Y%m%d%H%M%S'))
                    os.rename(storage_object.udev_path, file_backup)
                else:
                    os.remove(storage_object.udev_path)

            storage_object.delete()
    return None

def delete_target(iqn):
    for i in current_targets():
        if i == iqn:
            rtplib.Target(rtplib.FabricModule('iscsi'), wwn=iqn).delete()

    return None

# ----- Guardar configuración en targetcli -----

def save_to_disk():
    os.system("echo -e '\nsaveconfig\nY' | targetcli")

# ----- Main Function -----

if __name__ == '__main__':
    try:
        opts, args = getopt.getopt(sys.argv[1:], "bhm:", ["help"])
    except getopt.GetoptError:
        error_msg("Parametros incorrectos")

    # Iniciar variables de los argumentos
    master = ''
    backup = False

    # Recuperar variables de la linea de comandos
    for opt, arg in opts:
        if opt in ("-h", "--help"):
            print
            print __doc__
            sys.exit()

```



```

        elif opt in ('-m'):
            master = arg
        elif opt in ('-b'):
            backup = True

# Chequear parametros obligatorios
if not master:
    error_msg("Parametro -m obligatorio. Ejemplo -m master")

# Verificar parametro -m
if not check_master_exist(master):
    error_msg("Master no existente")

# Verificar iqn generado
iqn = IQN + master
if not rtplib.utils.is_valid_wwn('iqn',iqn):
    error_msg("%s no es un identificador wwn valido" % iqn)

# Iniciar repliegue
delete_target(iqn)
delete_fileio('_' + master,backup)

logging.info("[DELETEMASER] iscsi: " + iqn + " fileio: " + "_" + master + " file: "
+ master + ".img")

save_to_disk()
print "\n[ OK ] Master eliminado correctamente ver fichero log para ver detalles.\n"

```

tgtdeleteimage

```

#!/usr/bin/env python
# -*- coding: utf-8 -*-

```

ISCSI Target Manager - tgtdeleteimage

tgtdeleteimage. Utilizar para replegar imagenes target iSCSI y eliminar su imagen asociada del sistema de archivos

Opciones:

```

-i name      naming-authority a eliminar imagenes recursivamente o
             naming-authority y unique name para eliminar una imagen

-b          Crear un backup de la imagen master.

-t          Modo test. Visualiza imagenes sin eliminarlas

-h          Muestra la ayuda

```

Ejemplo:

```

tgtdeleteimage -i xx.yy -b
tgtdeleteimage -i xx.yy.zz:unique_name -b

```

```

import logging
import ConfigParser
import getopt
import sys
import os
import rtplib
import datetime

```

```

# ---- Recuperar datos del fichero de configuracion ----

```

```

FILE_TARGETMGR= '/etc/targetmgr/targetmgr.conf'

```

```

config = ConfigParser.ConfigParser()
config.read(FILE_TARGETMGR)

```

```

SERVER = config.get('SERVER', 'server')

```

```

PORT = config.get('SERVER', 'port')

PATH_LOG = config.get('PATH', 'path_log')
PATH_MASTER = config.get('PATH', 'path_master')
FILE_LOG = PATH_LOG + 'targetmgr.log'

IQN = config.get('IQN', 'iqn_prefix',) + "."

# ---- Crear logger ----
FORMAT_DATA='%Y-%d-%m %H:%M:%S'
FORMAT='% (asctime)s %(message)s'
logging.basicConfig(filename=FILE_LOG,format=FORMAT,level=logging.INFO,datefmt=FORMAT_DATA)

# ----- Funciones control de parametros -----

def error_msg(str_error):
    print "\nERROR: " + str_error + "\n"
    sys.exit(1)

def check_iqn(iqn):
    return rtplib.utils.is_valid_wwn('iqn',iqn)

# ----- iSCSI Funciones borrar fileio y target -----

def current_targets():
    existing_targets = []
    for target in rtplib.FabricModule('iscsi').targets:
        existing_targets.append(target.wwn,)
    return existing_targets

def delete_target_fileio(iqn,name,backup,test):
    # list log
    log_iqn=[]
    log_file=[]
    log_fileio=[]

    # delete target
    for i in current_targets():
        if i.startswith(iqn):
            log_iqn.append(i)
            if test:
                print "delete target iqn: "+ i
            else:
                rtplib.Target(rtplib.FabricModule('iscsi'), wwn=i).delete()

    # delete fileio
    for backstore in rtplib.root.RTSRoot().backstores:
        for storage_object in backstore.storage_objects:
            if storage_object.name.startswith(name):
                log_fileio.append(storage_object.name)
                log_file.append(storage_object.udev_path)
                if backup:
                    file_backup = storage_object.udev_path.replace('.img', '.' +
datetime.datetime.now().strftime('%Y%m%d%H%M%S'))
                    if test:
                        print "rename file:", storage_object.udev_path, "to",
file_backup
                    else:
                        os.rename(storage_object.udev_path, file_backup)
                else:
                    if test:
                        print "delete file:", storage_object.udev_path
                    else:
                        os.remove(storage_object.udev_path)
                if not test:
                    storage_object.delete()

    if not test:
        for i in range(0,len(log_iqn)):
            logging.info("[DELETEIMAGE] iscsi: " + log_iqn[i] + " fileio: " +
log_fileio[i] + " file: " + log_file[i])

```

```

    return None

# ----- Guardar configuración en targetcli -----
def save_to_disk():
    os.system("echo -e '\nsaveconfig\nY' | targetcli")

# ----- Main Function -----
if __name__ == '__main__':
    try:
        opts, args = getopt.getopt(sys.argv[1:], "bthi:", ["help"])
    except getopt.GetoptError:
        error_msg("Parametros incorrectos")

    # Iniciar variables de los argumentos
    image = ''
    backup = False
    test = False

    # Recuperar variables de la línea de comandos
    for opt, arg in opts:
        if opt in ("-h", "--help"):
            print
            print __doc__
            sys.exit()
        elif opt in ('-i'):
            image = arg
        elif opt in ('-b'):
            backup = True
        elif opt in ('-t'):
            test = True

    # Chequear parametros obligatorios
    if not image:
        error_msg("Parametro -i obligatorio. Ejemplo -i xx.yy")

    # Actualizar iqn
    iqn = IQN + image

    # Iniciar repliegue
    delete_target_fileio(iqn, image, backup, test)

    if not test:
        save_to_disk()
        print "\n[ OK ] Imagenes eliminadas correctamente ver fichero log para ver
detalles.\n"

```

Anexo C. Scripts PHP

boot.php

```
<?php
header ("Content-type: text/plain");
echo "#!ipxe\n";

function title($name) {
    $total_length = 107;
    $name_length = strlen($name);
    $start = intval(($total_length - $name_length) / 2);
    $end = $total_length - $start - $name_length;
    $title = str_repeat("-", $start) . "[" . $name . "]" . str_repeat("-", $end);
    echo "item --gap -- {$title}\n";
}

$url = "http://{$_SERVER["SERVER_ADDR"]}:{$_SERVER["SERVER_PORT"]}";

echo "console --x 1024 --y 768 --picture {$url}fondo.png\n";
echo ":menu\n";
echo "menu iPXE Institut Puig Castellar\n";
echo "set menu-default exit\n";
echo "set menu-timeout 10000\n";

title("Authentication Menu");
echo "item --key 1 login (1) Authentication\n";
echo "item --key 2 exit (2) Exit iPXE and continue BIOS boot\n";
echo "choose --default \${menu-default} --timeout \${menu-timeout} selected && goto \${selected} || exit 0\n";

echo ":login\n";
echo "login\n";
echo "isset \${username} && isset \${password} || goto menu\n";
echo "params\n";
echo "param username \${username}\n";
echo "param password \${password}\n";
echo "chain --replace --autofree {$url}menu.php##params\n";
echo ":exit\n";
echo "echo Booting from local disks ... \n";
echo "exit 0\n";

?>
```

menu.php

```
<?php

require_once('params.php');

echo ":menu\n";
echo $header;

echo "item --gap\n";
title("Images user: " . $username);
echo "item --gap\n";

$j=1;

foreach ($entries as $i) {
    echo "item n" . $j . " {$i}\n";
    $j++;
}

echo "item --gap\n";
echo "item backtotop Back to top\n";
echo "item signin Logout\n";

echo $default;
```

```

$j=1;

foreach ($commands as $i) {
    echo ":n" . $j . "\n";
    echo "{$i}\n";
    $j++;
}

echo ":backtotop\n";
echo "goto menu\n";
echo ":signin\n";
echo "chain --replace --autofree {$url}boot.php\n";
?>

```

params.php

```

<?php

require_once('globals.php');
require_once('config.php');
require_once('functions.php');
require_once('auth.php');

echo "params\n";
echo "param username \${username}\n";
echo "param password \${password}\n";

?>

```

globals.php

```

<?php
header("Content-type: text/plain");
echo "#!ipxe\n";
if (!isset($_POST['username']) || !isset($_POST['password'])) {
    exit();
}
$username = $_POST['username'];
$password = $_POST['password'];
$url      = "http://{$_SERVER["SERVER_ADDR"]}:{$_SERVER["SERVER_PORT"]}";
$header   = "menu Select Image - Institut Puig Castellar\n";
$access   = null;
$index    = 0;
$entries  = array();
$commands = array();
$default  = "choose selected && goto \${selected} || exit 0\n";
$offset   = 20;
$config   = parse_ini_file('/etc/targetmgr/targetmgr.conf');
?>

```

config.php

```

<?php

function path_to_commands($name) {
    global $username;
    global $config;

    $command = "sanboot iscsi:" . $config['server'] . "::<" . $config['port'] . "::<" .
$config['iqn_prefix'] . ".";

    $name = str_replace('.img','', $name);

    if ($username=="master") {
        $path_images = $config['path_master'];
        $name = str_replace($path_images,"", $name);
        $command = $command . $config['iqn_naming_master'] . ':'.$name;
    }
    else {
        $path_images = $config['path_images'];
    }
}

```

```

    $name = str_replace($path_images, "", $name);
    $name = str_ireplace('_', ':', $name);
    $command = $command . str_ireplace('/', '.', $name);
}

return $command;
}

function path_to_entries($name, $path) {
    global $username;
    global $config;

    $entry = "-> ";

    if ($username=='master') {
        $path_images = $config['path_master'];
        $path='';
    } else {
        $path_images = $config['path_images'];
    }

    $path = str_replace($path_images, "", $path);
    $name = str_replace('.img', "", $name);
    $name = str_replace('-', '.', $name);

    if ($username!="master") {
        $name = str_replace('_'.$username, "", $name);
        $path = str_ireplace('/', ' - ', $path) . ' ' . ' ';
        $path = str_pad(' ' . strtoupper($path), 60, '.') . ' > ';
    }

    return $entry . $path . parse_entry($name);
}

function parse_entry($str) {
    $out = "";
    $strlen = strlen($str);

    for ($i = 0; $i <=$strlen; $i++) {
        $char = substr( $str, $i, 1);

        if (ctype_upper($char)) {
            $out = $out . ' ' . $char;
        } elseif ((ctype_digit($char)) && ($i > 0)) {
            if (ctype_alpha(substr($str, $i-1, 1)))
                $out = $out . ' ' . $char;
            else
                $out = $out . $char;
        } else {
            $out = $out . $char;
        }
    }
    return $out;
}

if ($username == 'master')
    $directory=$config['path_master'];
else
    $directory=$config['path_images'];

$rri = new RecursiveIteratorIterator(new RecursiveDirectoryIterator($directory));

foreach ($rri as $file) {
    if ($file->isDir()) {
        continue;
    }

    if ((strcmp(substr(strrchr($file, "."), 1), "img") == 0)
        && (strpos($file, $username)!=false)){

        $commands[] = path_to_commands($file->getPathname());
        $entries[] = path_to_entries($file->getFilename(), $file->getPath());
    }
}

```

```
}  
?>
```

functions.php

```
<?php  
function authenticated() {  
    global $username;  
    global $password;  
    global $config;  
  
    $authenticated = False;  
  
    if ($username == 'master' && $password == $config['pwd_master']) {  
        $authenticated = True;  
    } else {  
        $handle = fopen("/etc/targetmgr/users", "r");  
  
        if ($handle) {  
            while (( $line = trim(fgets($handle))) != false) {  
                if (!empty($line)) {  
                    $user=explode(":",$line);  
                    if ((strcmp($user[0],$username)==0) && (strcmp($user[1],$password)==0)) {  
                        $authenticated = True;  
                        break;  
                    }  
                }  
            }  
            fclose($handle);  
        } else {  
            echo "error abriendo el fichero: users";  
        }  
    }  
    return $authenticated;  
}  
  
function title($name) {  
    # the max number of characters for resolution 1024 x 768 is 107  
    $total_length = 107;  
    $name_length = strlen($name);  
    $start = intval(($total_length - $name_length) / 2);  
    $end = $total_length - $start - $name_length;  
    $title = str_repeat("-", $start) . '[' . $name . ']' . str_repeat("-", $end);  
    echo "item --gap -- {$title}\n";  
}  
  
?>
```

auth.php

```
<?php  
if (!authenticated()) {  
    echo "echo Authentication failed!\n";  
    echo "sleep 3\n";  
    echo "chain --replace --autofree {$url}boot.php\n";  
    exit();  
}  
?>
```