# LFS : Lab 3 - Constrution d'un operating system avec ostree

| | |
|---|---|
| **Objet** | Constrution d'un operating system |
| **Niveau requis** | débutant, avisé |
| **Features** | |
| **Suivi** | :DONE: |

L'objectif de ce tutoriel est de construire un operating system avec les paquets préparés par la TOOLBOX et en utilisant OStree comme support de déploiement

OSTree est un système de gestion des versions des mises à jour des systèmes d'exploitation Linux. Il peut être considéré comme “Git pour les binaires du système d'exploitation”. Il fonctionne dans l'espace utilisateur et fonctionnera sur n'importe quel système de fichiers Linux.

Dans ce tutoriel l'approche consiste à stocker les composants dans des commits individuels dans un référentiel séparé, puis à les associer et à recréer un commit global.

Les paquets stockés dans les commits individuels (qu'il s'agisse de RPM, de fichiers zip ou autre) doivent être le résultat de make install DESTDIR= ou similaire.

# Présentation du système

Un système minimal recquiert essentiellement les paquets suivants (**220 Mo**):

- **linux**: le noyau linux
- **glibc**: la plupart des programmes qui tournent sous Linux utilisent des fonctions issues de la bibliotheque standard C GNU, glibc
- **Ncurses**: contient les bibliothèques de gestion des écrans type caractère, indépendant des terminaux.
- **BusyBox**: contient les versions réduites d'utilitaires UNIX courants compilés en un seul petit exécutable, ce qui fait de BusyBox une base idéale pour les systèmes à ressources limitées.
- **Gawk**: contient des programmes de manipulation de fichiers texte.
- **DropBear**: est un petit serveur et client SSH et est utile pour permettre l'accès shell distant à votre système.
- **getty et login**: (fournis par Busybox ou les programmes agetty et login fournis par le paquet util-linux) dont on a besoin pour pouvoir se connecter et obtenir l'invite de commande de bash
- **stty**: (fournit par Busybox ou CoreUtils) est utilisé dans `/etc/init.d/rc`, lui-même utilisé pour changer de niveau d'exécution et entrer dans le niveau initial

Cependant le système déployé dans ce tutoriel inclus beaucoup plus de paquets (**1,3 Go**):

1. Le noyau,la bibliothèque glib et les utilitaires de bases
2. les paquets de la LSB

3. systemd et ses dépendances
4. des utilitaires des systèmes de fichiers
5. des utilitaires de réseau
6. des utilitaires de compression
7. des utilitaires de partage réseau (curl, wget, git, rsync)
8. des languages de programmation (python, perl, java)

# Système de base

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 01 | linux | LFS | 80 Mo | Noyau linux | |
| 02 | glibc | LFS | 61 Mo | bibliothèque C principale fournissant toutes les routines basiques pour allouer de la mémoire, rechercher des répertoires, ouvrir et fermer des fichiers, les lire et les écrire, gérer les chaînes, faire correspondre des modèles, faire de l'arithmétique et ainsi de suite. | |
| 03 | busybox-hl | | 5 Mo | implémente un grand nombre des commandes standard sous Unix | |
| 04 | dropbear | | 5 Mo | client et serveur SSH | |
| 05 | kexec-tools | | 220 Ko | outil permettant de rebooter une machine sans passer par toute la couche hardware. C'est à dire qu'il va couper tous les services, descendre les init (sysV) pour arriver au bootloader. Ensuite il démarrera normalement et nous nous serons alors affranchit de la partie reboot hardware. | |
| 06 | kbd | | 36 Mo | contient les fichiers de tables de caractères, les polices de la console et des outils pour le clavier. | |
| 07 | rootfs | | 50 Ko | Système de fichier minimal | |
| | | | 187.27 Mo | | |

# LSB

La **L**inux **S**tandard **B**ase (LSB) est un projet joint par nombre de distributions Linux sous la structure organisationnelle du Free Standards Group afin de concevoir et standardiser la structure interne des systèmes d'exploitation basés sur GNU/Linux.

Les paquets suivants sont requis pour satisfaire les exigences LSB:

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 08 | at | BLFS | 1.6 Mo | permet de faire de l'exécution retardée et du traitement par lot. Il est requis pour être conforme à la LSB (Linux Standards Base). | |
| 09 | bash | LFS | 9,4 Mo | shell Bourne-Again. | readline |
| 10 | readline | LFS | 15 Mo | ensemble de bibliothèques qui offrent des fonctionnalités d'édition de la ligne de commande et d'historique. | |

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 11 | bc | LFS | 2.9 Mo | langage de traitement des nombres à la précision de votre choix. | |
| 12 | binutils | LFS | 52 Mo | contient un éditeur de liens, un assembleur et d'autres outils pour gérer des fichiers objets. | |
| 13 | coreutils | LFS | 17 Mo | outils pour afficher et configurer les caractéristiques basiques d'un système. | |
| 14 | cpio | BLFS | 17 Mo | contient les outils d'archivage. | |
| 15 | diffutils | LFS | 36 Mo | contient les programmes montrant les différences entre fichiers ou répertoires. | |
| 16 | vim | LFS | 38 Mo | LSB recquiert ED mais VIM remplace avantageusement cet éditeur de texte | |
| 17 | file | LFS | 20 Mo | outil pour déterminer le type d'un fichier ou des fichiers donnés. | |
| 18 | findutils | LFS | 2.6 Mo | contient les programmes de recherche de fichiers. Ces programmes sont fournis pour rechercher récursivement dans une hiérarchie de répertoires et pour créer, maintenir et chercher dans une base de données (souvent plus rapide que la recherche récursive mais moins fiable si la base de données n'a pas été mise à jour récemment). | |
| 19 | gawk | LFS | 12 Mo | contient les programmes de manipulation de fichiers texte. | |
| 20 | grep | LFS | 39 Mo | contient les programmes de recherche à l'intérieur de fichiers. | |
| 21 | gzip | LFS | 20 Mo | contient les programmes de compression et décompression de fichiers. | |
| 22 | m4 | LFS | 33 Mo | processeur de macros. | |
| 23 | man-db | LFS | 40 Mo | programmes pour trouver et voir des pages de manuel. | |
| 24 | ncurses | LFS | 41 Mo | bibliothèques de gestion des écrans type caractère, indépendant des terminaux. | |
| 25 | nspr | BLFS | 10 Mo | Netscape Portable Runtime (NSPR) offre une API indépendante de la plate-forme pour des fonctions au niveau système et de type libc. | |

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 26 | nss | BLFS | 8.8 Mo | Network Security Services (services de sécurité réseau) (NSS) est un ensemble de bibliothèques conçues pour supporter le développement en plate-forme croisée d'applications et de serveurs sécurisés. Les applications construites avec NSS peuvent supporter SSL v2 et v3, TLS, les certificats PKCS #5, PKCS #7, PKCS #11, PKCS #12, S/MIME, X.509 v3 et d'autres standards de sécurité. C'est utile pour implémenter SSL et S/MIME ou d'autres standards de sécurité sur Internet dans une application. | |
| 27 | Linux-PAM | BLFS | 26 Mo | Pluggable Authentication Modules (modules d'authentification connectables). C'est utile pour permettre à l'administrateur système local de choisir la façon dont s'authentifient les utilisateurs des applications. | |
| 28 | pax | BLFS | 920 Ko | contient les utilitaire d'archivage créé par POSIX et défini par le standard POSIX.1-2001. Plutôt que de trier les options incompatibles qui se sont glissées entre tar et cpio, avec leurs implémentations dans différentes versions d'UNIX, IEEE a conçu un nouvel utilitaire d'archivage. | |
| 29 | procps | LFS | 17 Mo | contient les programmes pour surveiller les processus. | |
| 30 | psmisc | LFS | 4.6 Mo | contient les programmes pour afficher des informations sur les processus en cours d'exécution. | |
| 31 | sed | LFS | 21 Mo | éditeur de flux. | |
| 32 | sendmail | BLFS | 14 Mo | agent de transport de courrier (MTA). | client-openldap,cyrus-sasl,openssl |
| 33 | openldap | BLFS | 49 Mo | fournit une implémentation libre de Lightweight Directory Access Protocol (protocole d'accès au répertoire). | |

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 34 | cyrus-sasl | BLFS | 26 Mo | contient une **S**imple **A**uthentication and **S**ecurity **L**ayer (simple couche d'authentification et de sécurité), une méthode pour ajouter le support de l'authentification aux protocoles basés sur la connexion. Pour utiliser **SASL**, un protocole se compose d'une commande d'identification et d'authentification d'un utilisateur sur un serveur ainsi que d'une négociation éventuelle de la protection des interactions consécutives du protocole. Si son utilisation est négociée, une couche de sécurité est insérée entre le protocole et la connexion. | |
| 35 | openssl | LFS | 46 Mo | contient des outils et des bibliothèques de gestion en matière de cryptographie. Ils servent à fournir des fonctions cryptographiques à d'autres paquets, comme OpenSSH, des applications de messagerie électronique et des navigateurs Internet (pour accéder à des sites HTTPS). | |
| 36 | shadow | LFS | 6.3 Mo | contient les programmes de gestion de mots de passe d'une façon sécurisée. | |
| 37 | tar | LFS | 45 Mo | programme d'archivage. | |
| 38 | time | BLFS | 4.0 Mo | programme qui mesure plusieurs des ressources CPU, comme le temps et la mémoire, que les autres programmes utilisent. La version GNU peut formater la sortie arbitrairement en utilisant une chaîne de formatage du style de printf pour inclure les nombreuses mesures des ressources. | |
| 39 | util-linux | LFS | 289 Mo | contient différents outils. Parmi eux se trouvent des outils de gestion des systèmes de fichiers, de consoles, de partitions et des messages. | |
| 40 | zlib | LFS | 5.1 Mo | routines de compression et décompression utilisées par quelques programmes. | |
| | | | 398.20 Mo | | |

# Systemd

systemd recquiert l'installation des paquets suivant :

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 41 | systemd | LFS | 32 Mo | contient des programmes pour contrôler le démarrage, l'exécution et l'arrêt du système. | |
| 42 | dbus | LFS | 18 Mo | système de bus de messages, une manière simple pour les applications de se parler. D-Bus fournit un démon système (pour les événements comme « l'ajout de nouveaux matériels » ou le « changement de la file d'impression ») et un démon individuel à chaque utilisateur connecté (pour les besoins généraux de communication entre les processus des applications de l'utilisateur). De plus, le bus des messages est construit sur la base d'un environnement de circulation des messages par communication directe, ce qui peut être utilisé par deux applications pour communiquer directement (sans passer par le démon de bus de messages). | expat |
| 43 | expat | LFS | 11 Mo | contient une bibliothèque C orientée flux pour analyser de l'XML. | |
| 44 | libcap | LFS | 8.5 Mo | implémente les interfaces du niveau utilisateur avec les fonctions POSIX 1003.1e disponibles dans les noyaux Linux. Ces possibilités établissent le partage des pouvoirs avec les privilèges root dans un ensemble de droits distincts. | |
| 45 | tcp_wrappers | LFS | 1.09 Mo | fournit des programmes d'enveloppes de démon qui affichent le nom du client qui interroge les services réseau et le service interrogé. | libnsl, rpcsvc-proto et libtirpc |
| 46 | libnsl | BLFS | 9.3 Mo | contient l'interface cliente publique de NIS(YP) et NIS+. Il remplace la bibliothèque NIS qui était présente dans glibc. | |
| 47 | rpcsvc-proto | BLFS | 2.6 Mo | contient les fichiers et les en-têtes rcpsvc protocol.x, précédemment inclus dans glibc et qui ne sont pas inclus dans le paquet libtirpc-1.2.5 de remplacement, ainsi que le programme rpcgen. | |
| 48 | libtirpc | BLFS | 8.3 Mo | contient des bibliothèques qui supportent des programmes utilisant l'API de Remote Procedure Call (RPC). Il remplace le RPC, mais pas les entrées de la bibliothèque NIS qui se trouvaient dans glibc. | |
| 49 | acl | LFS | 6.4 Mo | contient des outils d'administration des Access Control Lists (listes de contrôle d'accès) qui sont utilisés pour définir plus finement des droits d'accès de votre choix aux fichiers et aux répertoires. | |
| 50 | kmod | LFS | 13 Mo | contient des bibliothèques et des outils pour charger des modules du noyau | |

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 51 | libgcrypt | BLFS | 2.9 Mo | contient une bibliothèque de chiffrement à but généraliste basée sur le code utilisé dans GnuPG. La bibliothèque fournit une interface de haut niveau pour des composantes de chiffrement qui utilisent une API flexible et extensible. | libgpg-error |
| 52 | libgpg-error | BLFS | 9.9 Mo | contient une bibliothèque qui définit les valeurs habituelles d'erreur pour tous les composants de GnuPG. | |
| 53 | iptables | BLFS | 17 Mo | programme en ligne de commande et en espace utilisateur utilisé pour configurer l'ensemble de règles de filtrage de paquets des noyaux Linux 2.4 et supérieurs. | |
| 54 | lz4 | | 770 Ko | algorithme de compression de données sans perte. | |
| 55 | xz | LFS | 18 Mo | contient des programmes de compression et de décompression de fichiers. Il offre les possibilités des formats lzma et des formats de compression récents. La compression de fichiers textes avec xz donne un meilleur pourcentage de compression qu'avec les commandes gzip ou bzip2 traditionnelles. | |
| 56 | attr | LFS | 4.2 Mo | contient des outils d'administration des attributs étendus des objets du système de fichier. | |
| | | | 153.06 Mo | | |

# Systèmes de fichiers

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 57 | dosfstools | BLFS | 2.9 Mo | contient divers utilitaires pour les systèmes de fichiers de la famille FAT. | `CONFIG_MSDOS_FS` `CONFIG_VFAT_FS` |
| 58 | xfsprogs | BLFS | 5.3 Mo | contient des outils d'administration et de débogage pour le système de fichier XFS. | `CONFIG_XFS_FS` |
| 59 | zfs | | 20 Mo | système de fichiers open source sous licence CDDL. Les caractéristiques de ce système de fichiers sont sa très haute capacité de stockage, l'intégration de tous les concepts précédents concernant les systèmes de fichiers et la gestion de volume. | elfutils `CONFIG_ZFS` |
| 60 | elfutils | LFS | 3.5 Mo | bibliothèque pour gérer les fichiers ELF (Executable and Linkable Format). | |
| 61 | btrfs-progs | BLFS | 5.3 Mo | contient les outils d'administration et de débogage pour le système de fichier en B-arbre (btrfs). | `CONFIG_BTRFS_FS` `CONFIG_BTRFS_FS_POSIX_ACL` `CONFIG_REISERFS_FS_XATTR` |

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 62 | nfs-utils | BLFS | 20 Mo | contient le serveur en espace utilisateur et le client nécessaires pour utiliser les possibilités NFS du noyau. NFS est un protocole qui permet le partage de systèmes de fichiers sur un réseau. | CONFIG_NETWORK_FILESYSTEMS CONFIG_NFS_FS CONFIG_NFSD |
| 63 | libostree | | 1.6 Mo | libostree est à la fois une bibliothèque partagée et une suite d'outils de ligne de commande qui combine un modèle “git-like” pour valider et télécharger des arborescences de système de fichiers amorçables, ainsi qu'une couche pour les déployer et gérer la configuration du chargeur de démarrage. | gpgme, libassuan, glib, libxml2 |
| 64 | gpgme | LFS | 32 Mo | bibliothèque C qui permet d'ajouter le support du chiffrement à un programme. Il est conçu pour faciliter l'accès pour les applications à des moteurs de chiffrement de clés tels que GnuPG ou GpgSM. GPGME fournit une API de chiffrement de haut niveau pour le chiffrement, le déchiffrement, l'authentification, la vérification de signature et la gestion de clé. | libassuan |
| 65 | libassuan | LFS | 6.7 Mo | contient une bibliothèque de communication entre processus utilisée par certains des paquets liés à GnuPG. L'utilisation primaire de Libassuan est de permettre à un client d'interagir avec un serveur non permanent. Libassuan n'est toutefois pas limité à être utilisé avec des serveurs et des clients GnuPG. Il est conçu pour être suffisamment flexible pour correspondre aux demandes de la plupart des environnements basés sur de la transaction avec des serveurs non permanents. | libgpg-error |

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 66 | glib | LFS | 32 Mo | contient des bibliothèques de bas niveau utiles pour avoir la gestion de structures de données pour le C, des enveloppes de portabilité et des interfaces pour des fonctionnalités d'exécution telles qu'une boucle d'événements, les fils d'exécution, le chargement dynamique et un système d'objets. | |
| 67 | libxml2 | LFS | 13 Mo | contient des bibliothèques et des utilitaires utilisés pour analyser des fichiers XML. | |
| 68 | LVM2 | BLFS | 34 Mo | gère des partitions logiques. Il permet l'extension de systèmes de fichiers sur plusieurs disques physiques et plusieurs partitions de disque, il permet une navigation dynamique ou le bidouillage de partitions logiques. | CONFIG_MD CONFIG_BLK_DEV_DM CONFIG_DM_CRYPT CONFIG_DM_SNAPSHOT CONFIG_DM_THIN_PROVISIONING CONFIG_DM_MIRROR CONFIG_MAGIC_SYSRQ |
| 69 | squashfs | | 290 Ko | système de fichiers compressé en lecture seule sous Linux. Il peut être utilisé sur des supports de mémoire flash, CD-ROM ou disque dur pour des systèmes de fichiers disponible en lecture seulement. | CONFIG_SQUASHFS |
| | | | 176.59 | | |

# Network

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 70 | iproute2 | LFS | 14 Mo | contient des programmes pour le réseau, basique ou avancé, basé sur IPV4. | |
| 71 | bridge-utils | LFS | 916 Ko | contient un utilitaire nécessaire pour créer et gérer un périphérique de pont. Il est pratique dans l'initialisation d'un réseau pour une machine virtuelle (VM). | CONFIG_NET CONFIG_BRIDGE |
| 72 | openvswitch | | 35.4 Mo | implémentation open source d'un commutateur multicouche virtuel distribué. | CONFIG_OPENVSWITCH |
| | | | 50.31 Mo | | |

# Archivages

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 73 | lzo | BLFS | 12 Mo | bibliothèque de compression de données qui convient à la décompression et à la compression de données en temps réel. Cela signifie qu'elle favorise la vitesse et le ratio de compression. | |
| 74 | zstd | LFS | 16 Mo | algorithme de compression en temps réel qui fournit des ratios de compression élevés. Il propose une très large gamme de rapports entre compression et vitesse tout en étant soutenu par un décodeur très rapide. | |
| 75 | bzip2 | LFS | 6.4 Mo | contient des programmes de compression et décompression de fichiers. Compresser des fichiers texte avec bzip2 permet d'atteindre un taux de compression bien meilleur qu'avec l'outil gzip. | |
| 76 | libarchive | BLFS | 40 Mo | bibliothèque qui fournit une seule interface pour lire et écrire divers formats de compression. | |
| | | | 74.4 Mo | | |

# Téléchargements

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 77 | wget | BLFS | 60 Mo | contient un outil utile pour le téléchargement non interactif de fichiers issus du Web. | |
| 78 | rsync | BLFS | 11 Mo | contient l'outil rsync utile pour synchroniser de grosses archives de fichiers sur un réseau. | popt |
| 79 | popt | BLFS | 8 Mo | contient les bibliothèques popt qui sont utilisées par certains programmes pour analyser des options en ligne de commande. | |
| 80 | git | BLFS | 315 Mo | système de contrôle de versions distribué librement et open-source, conçu pour gérer du plus petit au plus gros projet rapidement et efficacement. Chaque clonage Git est un dépôt complet avec l'historique et les possibilités de poursuite des révisions, indépendamment de l'accès réseau ou d'un serveur central. Le système de branches et de synchronisation est rapide et facile à utiliser. | |
| 81 | curl-nss | BLFS | 77 Mo | contient un utilitaire et une bibliothèque utilisés pour le transfert de fichiers avec la syntaxe URL vers les protocoles suivants : FTP, FTPS, HTTP, HTTPS, SCP, SFTP, TFTP, TELNET, DICT, LDAP, LDAPS et FILE. Cette capacité de télécharger et de téléverser des fichiers peut être incorporée à d'autres programmes pour supporter des fonctions comme le streaming de média. | |
| | | | 471 Mo | | |

# Programmation

| Id | paquet | source | taille | Libellé | Dependances |
|---|---|---|---|---|---|
| 82 | Python-3 | BLFS | 220.9 Mo | contient l'environnement de développement Python. C'est utile pour la programmation orientée objet, l'écriture de scripts, le prototypage de gros programmes ou le développement d'applications entières. | |
| 83 | openjdk-8 | BLFS | 193.9 Mo | implémentation libre de la plateforme d'édition standard Java d'Oracle. OpenJDK est utile pour développer des programmes Java, et fournir un environnement d'exécution complet pour lancer des programmes Java. | |
| 84 | perl | BLFS | 75 Mo | contient le langage pratique d'extraction et de rapport (Practical Extraction and Report Language). | |
| | | | 489.9 Mo | | |

# Préparation du sytème

Le démarrage d'un système Linux se fait en 6 étapes. Donc, avant que la fenêtre d'identification apparaisse, 6 processus s'exécute à tour de rôle:

- 1ère étape : le BIOS. BIOS = Basic Input/Output system : système élémentaire d'entrée/sortie. ...
- 2e étape : la MBR. MBR = Master Boot Record : la zone amorce. ...
- 3e étape : le programme d'amorçage ou bootloader. ...
- **4e étape : le noyau. ...**
- 5e étape : init. ...
- 6e étape : Runlevel.

## Le noyau

Le noyau Linux chargé à l'étape 4 est de conception modulaire. On peut concevoir un noyau résident minimal qui sera chargé dans la mémoire. Par la suite, chaque fois qu'un utilisateur demande une fonction qui n'est pas présente dans le noyau résident, un module de noyau, parfois également appelé périphérique, devra être chargé en mémoire.

Cette manière de concevoir le noyau, nécessite d'utiliser un `initramfs` afin de pouvoir charger les modules nécessaires (essentiellement les drivers disques). Si un `initramfs` est utilisé, le processus de démarrage est plus long, même significativement plus long.

Lorsqu'on ne souhaite pas utiliser d'`initramfs`, il est nécessaire de compiler le noyau avec tous les drivers (modules) nécessaires compilés dans la partie résidente.

L'un des driver absolument indispensable pour pouvoir démarrer un distribution GNU/Linux est le driver de disque dur.

**Afin d'identifier le driver d'un périphérique SATA procédéder ainsi:**

- Récupérer l'abre complet du périphérique cible (dans l'exmple /dev/sdb)

```
ls -l /sys/block/sdb

lrwxrwxrwx. 1 root root 0  1 déc.  08:23 /sys/block/sdb ->
../devices/pci0000:00/0000:00:1c.0/0000:01:00.0/host
4/target4:1:3/4:1:3:0/block/sdb
```

- Descendre dans l'arbre pour repérer une ligne identifiant le driver de bas niveau.

Par exemple le lien sous `4:1:3:0` n'identifie pas un driver bas niveau, mais le driver générique **sd**:

```
ls -l /sys/devices/pci0000:00/0000:00:1c.0/0000:01:00.0/host
4/target4:1:3/4:1:3:0

lrwxrwxrwx    1 root     root             0 Jul  2 10:14 driver ->
../../../../bus/scsi/drivers/sd
```

Alors que le lien sous `0000:01:00.0` identifie le driver de bas niveau MPTSAS

```
ls -l /sys/devices/pci0000:00/0000:00:1c.0/0000:01:00.0

lrwxrwxrwx.  1 root root       0  1 déc.  08:23 driver ->
../../../../bus/pci/drivers/mptsas
```

- activer le driver ainsi relevé en kernel et non en module:

Afin que l'on puisse accéder aux disques gérés par ce driver lors du démarrage,

```
  [*] Fusion MPT device support  --->
    <*>   Fusion MPT ScsiHost drivers for SAS
```

# L'environnement de construction

L'OS doit intégrer les ressources physiques, par exemple le serveur de dev dispose de 2 disques physiques **sda** et **sdb**:

- le premier disque est réservé aux déploiements des distributions GNU/Linux construites à partir de l'environnement de compilation
- l'environnement de compilation a été déployé sur l'unique partition du deuxième disque `/dev/sdb1`

  Pour passer dans l'environnement de compilation il suffit de monter le deuxième disque, puis de chrooter dans celui-ci:

```
mount -o subvol=home /dev/sdb1 /mnt
cd /mnt/lfs/lfs
mount --bind /dev dev
mount --bind /proc proc
mount --bind /sys sys
chroot . /tools/bin/env -i \
    HOME=/root                  \
    TERM="$TERM"                \
    PS1='(lfs chroot) \u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin \
    /tools/bin/bash --login +h
```

ne pas oublier de contrôler, et si nécessaire mettre à jour la date du serveur, car c'est une condition bloquante pour pouvoir compiler en GCC

```
date --set "2020-07-06 15:48"
```

## Paramètres de la box

### Interface ETH1

- **addresse IPV4** : xx.xx.xxx.xxx
- **gateway** : xx.xx.xxx.x

### serveurs de noms

le fichier /etc/resolv.conf doit avoir le contenu suivant:

```
# Début de /etc/resolv.conf

nameserver 10.154.59.104
nameserver 10.156.32.33

# Fin de /etc/resolv.conf
```

# Construction d'un système de fichier

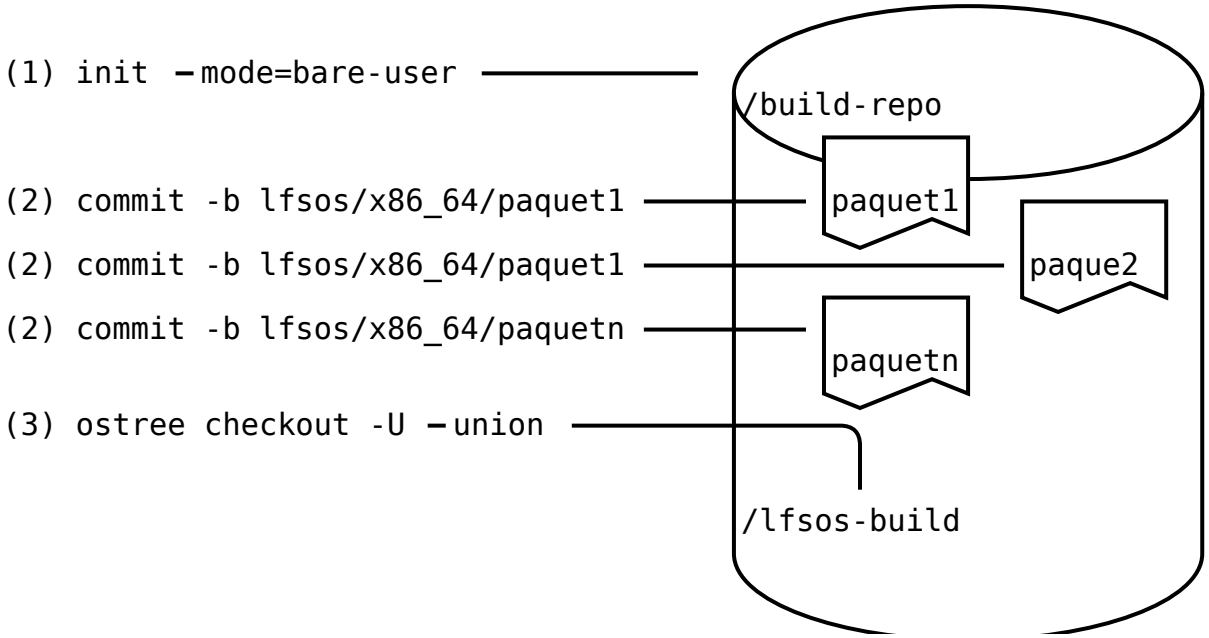
La construction d'un système de fichier avec **OSTree** repose essentiellement sur tois étapes

1. Etape 1: la construction d'un référenteil de paquets
2. Etape 2: la préparation du système de fichier sur lequel seront déployés les paquets

3. Etape 3: la construction de l'abre final par le déploiement des paquets dans le système de fichiers:

# Etape 1: Construction du référentiel de paquets

On va construire référentiel nu dans lequel on exporte le contenu des archives dans des branches séparées, puis un système de fichier dans le répertoire `/lfsos-build`

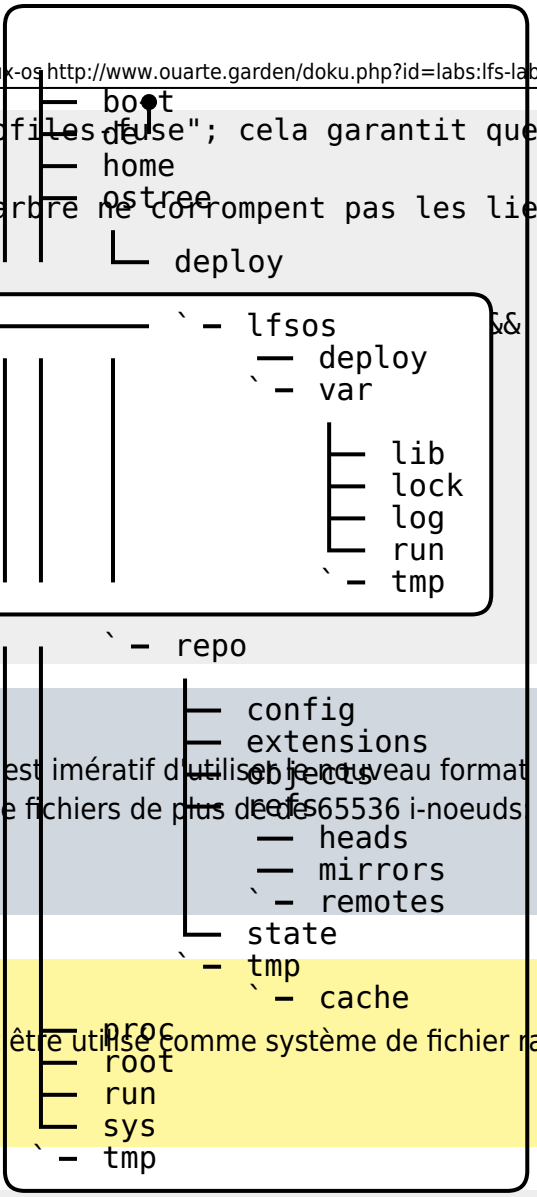



- (1) création d'un dépôt nu (/build-repo) –mode=bare-user
- (2) commit des paquets en tant que branches individuelles
- (3) **ostree checkout -U –union**: confection d'un système de fichie par l'union des paquets individuels

```
TIMEFORMAT='(BUSYBOX) Construction du tree en %R seconds ...'
time {
rm -Rf ${LFS}/build-repo &&
ostree --repo=${LFS}/build-repo init --mode=bare-user &&
mkdir -pv ${LFS}/build-repo  &&
pkglist=""
for pkg in $(echo -e "linux\nglibc\nbusybox-hl\ndropbear\nkexec-
tools\nkbd\nrootfs\nat\nbash\nreadline\nbc\nbinutils\ncoreutils\ncpio\ndiffu
tils\nvim\nfile\nfindutils\ngawk\ngrep\ngzip\nm4\nman-
db\nncurses\nnspr\nnss\nLinux-PAM\npax\nprocps-
ng\npsmisc\nsed\nsendmail\nclient-openldap\nopenssl\ncyrus-
sasl\nshadow\ntar\ntime\nutil-
linux\nzlib\nsystemd\ndbus\nexpat\nlibcap\ntcp_wrappers\nlibnsl\nrpcsvc-
proto\nlibtirpc\nacl\nkmod\nlibgcrypt\nlibgpg-
error\niptables\nlz4\nxz\nattr\ndosfstools\nxfsprogs\nzfs\nelfutils\nbtrfs-
progs\nnfs-
utils\nlibostree\ngpgme\nlibassuan\nglib-2\nlibxml2\nLVM2\nsquashfs\niproute
2\nbridge-
```

```
utils\nopenvswitch\nlzo\nzstd\nbzip2\nlibarchive\nwget\nrsync\npopt\ngit\ncu
rl-nss\nPython-3\nopenjdk-8\nperl"); do pkglist="${pkglist}$(git ls-remote -
-heads --tags http://user:password@xx.xx.xxx.xxx:/jacques/pkg.git | grep -E
"refs/tags/$pkg"|awk '{print $NF ; exit}'| cut -d '/' -f3)\\n" ; done
echo -e "$pkglist" > lfsos-pkg-list &&
rm -rf ${LFS}/lfsos-build &&
mkdir ${LFS}/lfsos-build &&
cd ${LFS}/lfsos-build &&
mkdir -pv ./{bin,boot,etc/{opt,sysconfig},home,lib/firmware,mnt,opt} &&
mkdir -pv ./{media/{floppy,cdrom},sbin,srv,var} &&
install -dv -m 0750 ./root &&
install -dv -m 1777 ./tmp ./var/tmp &&
mkdir -pv ./usr/{,local/}{bin,include,lib,sbin,src} &&
mkdir -pv ./usr/{,local/}share/{color,dict,doc,info,locale,man} &&
mkdir -v  ./usr/{,local/}share/{misc,terminfo,zoneinfo} &&
mkdir -v  ./usr/libexec &&
mkdir -pv ./usr/{,local/}share/man/man{1..8} &&
mkdir -v  ./usr/lib/pkgconfig &&
mkdir -v ./var/{log,mail,spool} &&
mkdir -pv ./var/{opt,cache,lib/{color,misc,locate},local} &&
touch ./var/log/{btmp,lastlog,wtmp} &&
chgrp -v utmp ./var/log/lastlog &&
chmod -v 664  ./var/log/lastlog &&
chmod -v 600  ./var/log/btmp &&
cd "${LFS}/" &&
for package in `cat lfsos-pkg-list`; do
ostree --repo=${LFS}/build-repo commit -b lfsos/x86_64/${package} --
tree=tar=${LFS}/pkg/${package}.tar.gz;
done &&
for package in `cat lfsos-pkg-list`; do
ostree --repo=${LFS}/build-repo checkout -U --union lfsos/x86_64/${package}
lfsos-build;
done &&
cd /lfsos-build &&
if [ -d ./var/run ]; then
  # Take action if $DIR exists. #
  for DIR in `find ./var/run -mindepth 1 -type d`
  do
    #Do whatever you need with DIR
    install -d ./run/$(echo ${DIR} |sed 's|.*/||')
  done
  rm -rf ./var/run
fi &&
ln -sv /run ./var/run &&
ln -sv /run/lock ./var/lock &&
for file in $(ls ./etc/systemd/network/10-eth*-static*); do vim -c 'set
title titlestring=...'$file'' $file; done &&
vim -c 'set title titlestring=.../etc/resolv.conf' ./etc/resolv.conf &&
# Changement du mot de passe dans /etc/shadow
sed -i '/^'root'/s/\([^:]*\):[^:]*:\(.*\)/\1:'"$(echo 'password' |openssl
passwd -1 -stdin | cut -d" " -f1)"':\2/' ./etc/shadow
```

```
# Mise en place d'un point de montage "rofiles-fuse"; cela garantit que tous
les processus
# exécutés pour le post-traitement de l'arbre ne corrompent pas les liens
physiques.
install -d ${LFS}/lfsos-mnt &&
mount -o bind ${LFS}/lfsos ${LFS}/lfsos-mnt &&

ostree admin init-fs ${LFS}/lfsos-mnt &&

ostree admin os-init lfsos &&

ostree admin --sysroot=${LFS}/lfsos-mnt os-init lfsos &&

umount ${LFS}/lfsos-mnt
```




pour avoir une image du sytème complet, il est imératif d'utiliser le nouveau format portable (SVR4), qui accepte les systèmes de fichiers de plus de de 65536 i-noeuds `-H newc

Le système créé dans l'abre transitoire peut être utilisé comme système de fichier racine sur une partition.

```
mount -t btrfs /dev/sda2 /mnt
cd /mnt
btrfs subvolume create root
cd ../
umount /mnt
mount -o subvol=root /dev/sda2 /mnt
rsync -rav /lfsos-build/ /mnt
```

- (4) **ostree admin init-fs** Initialise un système de fichiers racine physique vide dans le CHEMIN désigné, avec des niveaux supérieurs normaux et des autorisations correctes pour chaque répertoire. Principalement utile pour les installateurs de systèmes d'exploitation.
- (5) **ostree admin os-init OSNAME** Initialise un nouvel état pour un système d'exploitation. Garantit que les sous-répertoires principaux de `/var` (`/tmp`, `/lib`, `/run` et `/lock`) existent et initialise le OSNAME donné en tant que racine OSTree. Chaque emplacement de déploiement est composé d'une seule `/var` partagée et d'un ensemble de déploiements (chroots).

## Etape 2: Initialisation du système de fichiers

Lorsqu'on déploie un os, **OSTree** vérouille le répertoire dans lequel l'os est déployé. Pour cette raison l'utilisation des outils classiques (rm, mv, ln) à l'intérieur du chroot peut être la source de disfonctionnement voire perturber le file système sous jacent (impossibilité de détruire un dossier). Pour cette raison on utilisera un disque virtuel. Comme cela implique de travailler dans un jail chrooté, le déploiement automatisé de GRUB échourea lors de l'exécution de **grub-probe**, il faut donc remplacer le version binaire de cet utilitaire par un faux exécutable dont l'unique fonction est de répondre positivement:

```
if [[ $(file /sbin/grub-probe ) =~ 'text' ]]; then
  exit;
else
  echo "faker script for grub-probe"
mv /sbin/grub-probe /sbin/grub-probe.orig &&
```

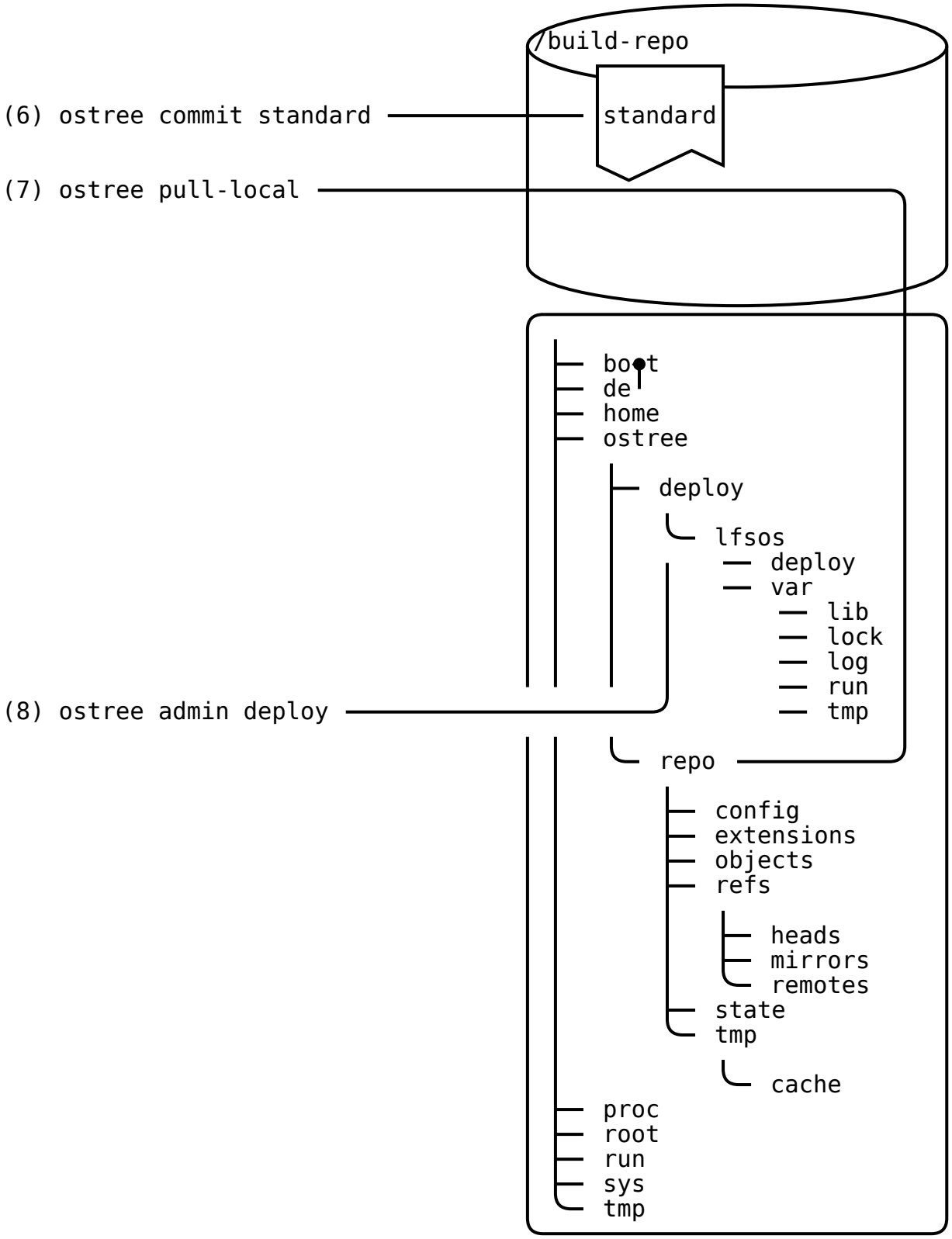
```
cat > /sbin/grub-probe << "EOF"
#!/bin/bash
#
# A faker script for grub-probe, if your's stops working properly
#

# Depends on /root/grub.hack file

# . /root/grub.hach

arg_is_device=0
target=""

# Call Order:
# / --target=device
# --target=fs_uuid /dev/sda1 --device
# /boot --target=device
# --target=fs_uuid /boot --device
# "/" "--target=fs"
# "--target=abstraction" "ext2" "--device"
# "--target=fs" "ext2" "--device"
# "--target=drive" "ext2" "--device"
# "--target=fs_uuid" "ext2" "--device"
# "--target=abstraction" "ext2" "--device"
# "--target=fs" "ext2" "--device"
# "--target=drive" "ext2" "--device"
# "--target=fs_uuid" "ext2" "--device"
# "/" "--target=device"
# "--target=fs_uuid" "ext2" "--device"
# "/boot" "--target=device"
# "--target=fs_uuid" "ext2" "--device"
# "/" "--target=fs"
# "--target=abstraction" "ext2" "--device"
# "--target=fs" "ext2" "--device"
# "--target=drive" "ext2" "--device"
# "--target=fs_uuid" "ext2" "--device"
# "--target=abstraction" "ext2" "--device"
# "--target=fs" "ext2" "--device"
# "--target=drive" "ext2" "--device"
# "--target=fs_uuid" "ext2" "--device"

for opt in $@
do
    case "${opt}" in
    --device)
        # Means the thing in $arg is a device
        arg_is_device=1
        ;;
    --target=*)
        # = device, = fs, = fs_uuid
        target=${opt#*=}
```

```
        ;;
    *)
        arg=$opt
        ;;
    esac
done

set >> /tmp/probe.env

case "$target" in
abstraction)
    echo
    ;;
device)
    echo "/dev/sda1"
    ;;
drive)
    echo "(hd0)"
    ;;
fs)
    # Returns the filesystem type
    echo "ext2"
    ;;
fs_uuid)
    tune2fs -l $arg | awk '/Filesystem UUID/ { print $3 }'
    ;;
esac
EOF

chmod 0755 /sbin/grub-probe
fi
```

```
TIMEFORMAT='(BUSYBOX) Deploiement image OSTree en %R seconds ...'
time {
qemu-img create -f qcow2 lfsos001.qcow2 8G &&
qemu-nbd -v -c /dev/nbd0 lfsos001.qcow2  &&
mv -v /lfsos-build/etc/* /lfsos-build/usr/etc
parted --script /dev/nbd0 \
    mklabel gpt \
    mkpart primary 1MiB 100% \
    set 1 esp on
mkfs.ext4 /dev/nbd0p1
mkdir -pv /tmp/lfsos001
mount /dev/nbd0p1 /tmp/lfsos001
cd /tmp/lfsos001
for rep in $(echo -e "bin\ndev\nlib\nlib64\nproc\nusr\nsbin\nsys\nlfsos-
build\nbuild-repo"); do
  mkdir -pv $rep
  mount --bind /$rep ./$rep
done
```

```
chroot .
ostree admin init-fs .
ostree admin os-init lfsos
}
```

# Etape 3: Construction de l'arbre final




- (6) **ostree commit standard**: créer un nouveau commit global dans la branche lfsos/x86_64/standard.
- (7) **ostree pull-local**: Copie les données d'un référentiel donné (dans le répertoire repo)
- (8) **ostree admin deploy**: met le nouveau déploiement par défaut au redémarrage.

```
TIMEFORMAT='(BUSYBOX) Deploiement image OSTree en %R seconds ...'
```

```
time {
checksum=$(ostree --repo=${LFS}/build-repo commit -b lfsos/x86_64/standard --link-checkout-speedup /lfsos-build) &&
ostree --repo=ostree/repo pull-local /build-repo lfsos/x86_64/standard &&
ostree admin deploy --os=lfsos --karg-proc-cmdline --karg=enforcing=0 lfsos/x86_64/standard &&
exit
for rep in $(echo -e "bin\ndev\nlib\nlib64\nproc\nusr\nsbin\nsys\nlfsos-build\nbuild-repo"); do
  umount ./$rep
done
rm -rf /ost-lfsos-9.0.gz
find . -print | cpio -o -H newc | gzip -9 > /ost-lfsos-9.0.gz &&
cd "${LFS}/"
}
```

Ne pas oublier de restaurer la binaire grub-probe

```
if [[ $(file /sbin/grub-probe ) =~ 'text' ]]; then
  echo "restauration grub-probe"
  mv /sbin/grub-probe.orig /sbin/grub-probe
fi
```

# Déploiement sur un disque

Au démarrage du système :

1. soit le micrologiciel **BIOS** charge les 512 premiers octets de ce disque, ces 512 octets constituant le **MBR** (master boot record ou zone d'amorçage) ou lorsque le disque de boot a plusieurs partitions, le micrologiciel BIOS lit le **MBR** du disque, puis le **VBR** de la partition (Volume Boot Record). À partir de ces informations, il peut déterminer l'emplacement du chargeur d'amorçage et le lancer.
2. soit le micrologiciel **UEFI** (et non pas le BIOS) est utilisé pour lancer le chargeur d'amorçage : l'UEFI lit la **GPT** du disque (GUID Partition Table) pour déterminer l'emplacement de la routine d'amorçage.

Quelque soit le cas il faut créer une partition dédiée au début du disque, car le changement de micrologiciel ne modifie pas le fait qu'on a toujours besoins d'un emplacement spécifique pour le bootloader:
- **BIOS** charge le bootloader depuis le MBR du disque (pour la plupart des utilisateurs, une partition de démarrage (boot) de 250 Mo est suffisante mais elle peut aller audelà).
- **UEFI** charge les fichiers placés dans la partition prévue à cette effet, l'ESP (doit être situé au début d'un disque GPT et avoir un indicateur “boot” et une Taille comprise entre

100 et 250 Mo)

# Préparation des disques

## Partitionnement du disque

Pour créer une table de partition GPT

```
parted --script /dev/sdb \
    mklabel gpt \
    mkpart primary 1MiB 261MiB \
    mkpart primary 261MiB 100% \
    set 1 esp on
```

Pour créer une table de partition MBR

```
parted --script /dev/sdb \
    mklabel msdos \
    mkpart primary 1MiB 261MiB \
    mkpart primary 261MiB 100% \
    set 1 esp on
```

blkid -s UUID -o value /dev/sdb2 2041109a-0a94-4683-94b2-0ef2e6c45a2c

## Formatage des partitions

La partition qui héberge le bootloader (BOOT ou ESP) doit être formatée en FAT32

```
mkfs.fat -F 32 /dev/sdb1
```

La partition qui va recevoir le système de fichier de la distribution doit être formatée dans l'une des formats gérés par le Noyau.

```
mkfs.btrfs /dev/sdb2
```

Le fichier `/etc/fstab` doit être adapté pour pointer vers le bon disque système.

```
cd /mnt
```

```
mv usr/etc .
cat > etc/fstab << "EOF"
# Begin /etc/fstab

# file system  mount-point  type     options             dump  fsck
#                                                              order

/dev/sda2      /            btrfs    subvol=/root,defaults,noatime   1     1
proc           /proc        proc     nosuid,noexec,nodev 0     0
sysfs          /sys         sysfs    nosuid,noexec,nodev 0     0
devpts         /dev/pts     devpts   gid=5,mode=620      0     0
tmpfs          /run         tmpfs    defaults            0     0
devtmpfs       /dev         devtmpfs mode=0755,nosuid    0     0

# End /etc/fstab
EOF
```

# Le chargeur de démarrage

Le noyau linux, le fichier System.map ainsi que la sauvegarde du fichier config utilisé pour compiler le noyau doivent être déposés à la racine de la partition **ESP**

```
mount /dev/sdb1 /boot/efi
cp -a ls /src/linux-5.4.8/_pkg/boot/* /boot/efi
```

Lorsque le noyau Linux prend en charge le démarrage **EFISTUB** cela permet au micrologiciel EFI de charger le noyau en tant qu'exécutable EFI, indirectement en utilisant un chargeur de démarrage ou directement par une carte mère UEFI.

Pour installer le gestionnaire de démarrage EFI, il faut s'assurer d'abord que le système a démarré en mode UEFI et que les variables UEFI sont accessibles. Cela peut être vérifié en exécutant la commande `efivar --list` ou, si **efivar** n'est pas installé, en faisant `ls /sys/firmware/efi/efivars` (si le répertoire existe, le système est démarré en mode UEFI).

lors de ce tutoriel deux chargeurs de démarrage ont été testés avec succès, **rEFInd** et **systemd-boot**.

### rEFInd

**rEFInd** Boot Manager graphique, fork actif de rEFIt

Pour Installer **rEFInd**, il faut de monter la partition dédiée sur `/boot/efi`, télécharger le binaire **refind-install** puis l'exécuter:

```
mount /dev/sdb1 /boot/efi
mkdir /boot/efi/EFI
cd /download/refind-bin-0.11.5/
./refind-install
ShimSource is none
Installing rEFInd on Linux....
ESP was found at /boot/efi using vfat
Copied rEFInd binary files

Copying sample configuration file as refind.conf; edit this file to
configure
rEFInd.

Could not parse device path: Invalid argument
Could not parse device path: Invalid argument
Creating new NVRAM entry
rEFInd is set as the default boot manager.
Existing //boot/refind_linux.conf found; not overwriting.
exit
```

Commme dans l'exemple il sera peut-être nécessaire de forcer l'insertion d'une entrée de démarrage avec **efibootmgr**.

```
efibootmgr -v -c -d /dev/sdb -p 1 -l \\EFI\\refind\\refind_x64.efi -L rEFInd

BootCurrent: 0008
Timeout: 10 seconds
BootOrder:
000E,0000,0001,0002,0003,0008,000C,0004,0005,0006,0007,0009,000A,000B,000D
Boot0000* CD/DVD Rom    ACPI(a0341d0,0)PCI(1f,2)ATAPI(0,0,0)CD-
ROM(1,1c8,30f8)
Boot0001* Floppy Disk   Vendor(0c588db8-6af4-11dd-a992-00197d890238,00)
Boot0002* Hard Disk 0
ACPI(a0341d0,0)PCI(1c,0)PCI(0,0)SCSI(3,0)HD(2,103000,5b800,b381ffad-
b370-4e67-9bc0-64032de09b5f)File(\EFI\grub\grubx64.efi)
Boot0003* PXE Network   ACPI(a0341d0,0)PCI(1,0)PCI(0,0)MAC(e41f1360e8e4,0)
Boot0004* Hard Disk 1   Vendor(0c588db8-6af4-11dd-a992-00197d890238,09)
Boot0005* Hard Disk 2   Vendor(0c588db8-6af4-11dd-a992-00197d890238,0a)
Boot0006* Hard Disk 3   Vendor(0c588db8-6af4-11dd-a992-00197d890238,0b)
Boot0007* Hard Disk 4   Vendor(0c588db8-6af4-11dd-a992-00197d890238,0e)
Boot0008* USB Storage
ACPI(a0341d0,0)PCI(1a,7)USB(1,0)HD(1,ac,2774,03c37324)
Boot0009* Diagnostics   Vendor(0c588db8-6af4-11dd-a992-00197d890238,da)
Boot000A* iSCSI Vendor(0c588db8-6af4-11dd-a992-00197d890238,04)
Boot000B* iSCSI Critical    Vendor(0c588db8-6af4-11dd-a992-00197d890238,05)
Boot000C* Legacy Only   Vendor(0c588db8-6af4-11dd-a992-00197d890238,ee)
```

```
Boot000D* Embedded Hypervisor   Vendor(0c588db8-6af4-11dd-
a992-00197d890238,01)
Boot000E* rEFInd    HD(1,800,82000,a64c020c)File(\EFI\refind\refind_x64.efi)
```

Pour terminer l'installation, configurer `refind.conf`.

```
cat >> /boot/efi/EFI/refind/refind.conf <<EOF
menuentry "LFSos" {
    icon EFI/refind/icons/os_linux.png
    volume 2041109a-0a94-4683-94b2-0ef2e6c45a2c
    loader
vmlinuz-4ca3a5f1c599a63b40e1d8ab2a8830f0e35b7908a230749c7a3b7e80253fa655
    options "root=UUID=2041109a-0a94-4683-94b2-0ef2e6c45a2c rw
rootflags=subvol=root"
}
EOF
```

## Systemd-boot

**Systemd-boot** est un simple gestionnaire de démarrage UEFI qui exécute des images EFI configurées. Il est inclus par défaut dans systemd.

Après avoir monté l'ESP sur `/boot/efi`, utiliser `bootctl` pour installer **systemd-boot** dans la partition système EFI en exécutant:

```
mount /dev/sdb1 /boot
bootctl install
```

Cela copiera le chargeur de démarrage **systemd-boot** sur la partition EFI: sur un système d'architecture x64, les deux fichiers binaires identiques `/boot/efi/EFI/systemd/systemd-bootx64.efi` et `/boot/efi/EFI/BOOT/BOOTX64.EFI` seront transférés vers l'ESP . Il définira ensuite **systemd-boot** comme l'application EFI par défaut (entrée de démarrage par défaut) chargée par le gestionnaire de démarrage EFI.

Commme pour rEFInd il sera peut-être nécessaire de forcer l'insertion d'une entrée de démarrage, en dehors du processus automatisé avec **efibootmgr**.

```
efibootmgr -v -c -d /dev/sdb -p 1 -l \\EFI\\systemd\\systemd-bootx64.efi` -L
bootctl
```

Pour terminer l'installation, configurer systemd-boot.

Le fichier `/boot/loader/loader.conf` doit pointer vers un menu dans le répertoire `/boot/loader/entries/`

```
cat > /boot/loader/loader.conf << "EOF"
#timeout 3
```

```
#console-mode keep
#default 262496a661004cb99e9b543ad12c5750-*
default lfsos
timeout 1
editor 0
EOF
```

Le fichier `/boot/loader/entries/xxxxxx.conf` est le fichier de boot qui sera exécuté:

```
cat > /boot/loader/entries/lfsos.conf <<EOF
title   lfsos
linux
/vmlinuz-75666a4cac006bf2ce21e2c0874eca8b9fc0152295a0bf2147f9927ada62bb7c
options root=/dev/sda2 rootflags=subvol=root rw
EOF
```

A l'issue de la configuration l'arbre de la partition doit ressembler à quelque chose comme ça:

```
/boot
├── 262496a661004cb99e9b543ad12c5750
├── EFI
│   ├── BOOT
│   │   └── BOOTX64.EFI
│   ├── Linux
│   └── systemd
│       └── systemd-bootx64.efi
└── loader
    ├── entries
    │   └── lfsos.conf
    ├── loader.conf
    └── random-seed

7 directories, 5 files
```