Linux From Scratch

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Table of Contents

Linux From Scratch

Version 5.0

Gerard Beekmans

Copyright © 1999−2003 Gerard Beekmans

Ce livre décrit le processus de création d'un système Linux depuis rien, en utilisant uniquement les sources des logiciels utilisés.

Copyright (c) 1999−2003, Gerard Beekmans

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions in any form must retain the above copyright notice, this list of conditions and the following disclaimer.
- Neither the name of "Linux From Scratch" nor the names of its contributors may be used to endorse or promote products derived from this material without specific prior written permission.
- Any material derived from Linux From Scratch must contain a reference to the "Linux From Scratch" project.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Dedication

Ce livre est dédié à ma femme aimante, *Beverly Beekmans*, au soutien indéfectible.

Table des matières

Préface

Avant−propos

Ayant utilisé certaines distributions Linux, je n'ai jamais été satisfait par aucune d'entre elles. Je n'aimais pas la façon dont les scripts de démarrage étaient arrangés. Je n'aimais pas la façon dont certains programmes étaient configurés par défaut. Beaucoup de choses comme celles−ci m'ennuyaient. Finalement, j'ai réalisé que si je souhaitais avoir une complète satisfaction de mon système Linux, je devrais construire mon propre système à partir de rien, en utilisant uniquement les codes source. Je me suis résolu à n'utiliser aucun package pré−compilé, aucun CD−Rom ou disque de démarrage qui aurait installé quelques utilitaires simples. J'utiliserai mon système Linux courant pour développer le mien.

Cette idée étrange semblait très difficile à ce moment et m'a souvent paru une tâche impossible. Après avoir traité toutes sortes de problèmes, comme les erreurs de dépendances et de compilation, un système Linux personnalisé et complètement opérationnel était créé. Je l'ai appelé Linux From Scratch (Linux Par Le Début, ou Linux à Partir de Rien) ou, plus simplement, LFS.

J'espère que vous prendrez plaisir à travailler sur votre LFS !

−− Gerard Beekmans gerard@linuxfromscratch.org

Audience

Qui voudrait lire ce livre

Il y a beaucoup de raisons qui pousseraient quelqu'un à vouloir lire ce livre. La raison principale est d'installer un système LFS. La question que beaucoup de personnes se posent est « pourquoi se fatiguer à installer manuellement un système Linux depuis le début alors qu'il suffit de télécharger une distribution existante ? ». C'est une bonne question.

Une raison importante de l'existence de LFS est d'apprendre comment fonctionne un système Linux de l'intérieur. Construire un système LFS vous apprend tout ce qui fait que Linux fonctionne, et comment les choses interagissent et dépendent les unes des autres, et le plus important, vous apprend à le personnaliser afin qu'il soit à votre goût et réponde à vos besoins.

Un avantage clé de LFS est que vous avez plus de contrôle sur votre système sans avoir à dépendre d'une implémentation créée par quelqu'un d'autre. Avec LFS, vous êtes maintenant sur le siège conducteur et êtes capable de décider chaque aspect de votre système, comme la disposition des répertoires ou la configuration des scripts de démarrage. Vous saurez également exactement où, pourquoi et comment les programmes sont installés.

Un autre avantage de LFS est la possibilité de créer un système Linux très compact. Quand vous installez une distribution courante, vous finissez par installer beaucoup de programmes que vous n'utiliserez jamais de votre vie. Ils sont juste là et occupent un espace disque précieux. Il n'est pas difficile de construire un système LFS de moins de 100 Mo. Cela vous semble−t−il toujours beaucoup ? Certains d'entre nous ont travaillé afin de créer un système LFS minuscule. Nous avons installé un système juste suffisant pour faire fonctionner le serveur web Apache; l'espace disque total occupé était approximativement de 8 Mo. Avec plus de dépouillement encore, cela peut être ramené à 5 Mo ou moins. Essayez donc d'en faire autant avec une distribution courante!

Si nous devions comparer une distribution Linux à un hamburger que vous achetez au restaurant fast−food, vous n'avez aucune idée de ce que vous mangez. LFS ne vous donne pas un hamburger, mais la recette pour faire un hamburger. Cela vous permet de prudemment l'inspecter, d'enlever les ingrédients non désirés, et par la même occasion vous permet de rajouter des ingrédients qui correspondent mieux à la saveur que vous attendez de ce hamburger. Quand vous êtes satisfait des ingrédients, vous passez à l'étape suivante en les combinant ensemble. Vous avez désormais la chance de pouvoir le faire de la façon dont vous le souhaitez: grillez−le, faites−le cuire au four, faites−le frire, faites−le au barbecue ou mangez−le cru.

Une autre analogie que nous pouvons utiliser est de comparer LFS à une maison construite. LFS vous donnera les murs de la maison, mais c'est à vous de la construire, en vous donnant la liberté d'ajuster vos plans comme vous le souhaitez.

Un autre avantage d'un système Linux personnalisé est un surcroît de sécurité. Vous compilerez le système complet à partir de la base, ce qui vous permet de tout vérifier, si vous le voulez, et d'appliquer tous les correctifs de sécurité que vous voulez ou devez appliquer. Vous n'avez pas à attendre que quelqu'un d'autre vous fournisse un package réparant une faille de sécurité. Malgré tout vous n'avez aucune garantie que le nouveau package résolve le problème de manière adéquate. Vous ne pourrez jamais savoir si une faille de sécurité est réparée si vous ne le faites pas vous−même.

Qui ne voudrait pas lire ce livre

Si vous ne souhaitez pas contruire votre propre système à partir des sources, alors vous ne voudrez probablement pas lire ce livre. Notre but est de construire les fondations d'un système complet et utilisable. Si vous souhaitez seulement connaître ce qui se passe lorsque votre ordinateur démarre, alors nous vous recommandons le guide pratique << From−PowerUp−To−Bash−Prompt >>. Ce guide pratique construit un système basique similaire à celui de ce livre, mais il s'occupe de la création d'un système capable de démarrer jusqu'au prompt du shell Bash.

Pendant que vous réfléchissez à celui que vous allez lire, cherchez votre objectif. Si vous souhaitez construire un système Linux tout en apprenant, alors ce livre est certainement le meilleur choix. Si votre objectif est strictement éducatif, et que vous n'avez aucune envie d'utiliser le système en question, alors le guide pratique << From−PowerUp−To−Bash−Prompt >> est probablement le meilleur choix.

Le guide pratique << From−PowerUp−To−Bash−Prompt >> est disponible sur http://axiom.anu.edu.au/~okeefe/p2b/.

Prérequis

Ce livre suppose que son lecteur possède un bon ensemble de connaissances sur l'utilisation et l'installation de systèmes Linux. Avant de commencer à construire votre système LFS, vous devriez lire les guides pratiques suivants :

- Software−Building−HOWTO

  C'est un guide complet sur la construction et l'installation « générique » de logiciels UNIX sous Linux. Ce guide pratique est disponible sur http://www.tldp.org/HOWTO/Software−Building−HOWTO.html.
- The Linux Users' Guide

Ce guide couvre l'utilisation de différents logiciels Linux et est disponible sur http://espc22.murdoch.edu.au/~stewart/guide/guide.html.

- L'astuce essentielle à lire avant tout

  C'est une astuce LFS écrite spécifiquement pour les nouveaux utilisateurs Linux. C'est principalement une liste de liens de sources excellentes d'informations sur une grande gamme de thèmes. Toute personne essayant d'installer LFS devrait au moins avoir une certaine compréhension de la majorité des thèmes de cette astuce. Elle est disponible sur http://www.linuxfromscratch.org/hints/downloads/files/essential_prereading.txt.

Structure

Ce livre est divisé en quatre parties :

Partie I − Introduction

La première partie donne quelques informations importantes, par exemple sur la façon d'installer, ainsi que sur le livre (journal des modifications, remerciements, listes de diffusion associées, et ainsi de suite).

Partie II − Préparation de la construction

La deuxième partie décrit comment préparer le processus de construction : création d'une partition, téléchargement des packages et compilation d'outils temporaires.

Partie III − Construction du système LFS

La troisième partie vous guidera à travers la construction du système LFS : compilation et installation de tous les packages un par un, configuration des scripts de démarrage et installation du noyau. Le système Linux résultant sera la fondation sur laquelle vous pourrez construire d'autres logiciels, pour étendre votre système de la façon dont vous le souhaiterez.

Partie IV − Annexes

La quatrième partie consiste en deux annexes. La première est une liste alphabétique de tous les packages installés, chaque package donnant son site officiel de téléchargement, son contenu et ses dépendances d'installation. La deuxième annexe liste tous les programmes et bibliothèques installés par ces packages en ordre alphabétique, de façon à ce que vous puissiez retrouver à quel package appartient un certain programme ou une certaine bibliothèque.

(Une grande partie de l'annexe A est intégrée aux parties II et III. Ceci augmente la taille du livre mais nous croyons que cela le rend plus facile à lire. Vous n'avez pas besoin de vous référer à l'annexe lors de l'installation. Ce va−et−vient serait gênant, et spécialement si vous utilisez la version texte du livre.)

I. Première Partie − Introduction

Table des matières

Chapitre 1. Introduction

Comment les choses se dérouleront−elles

Vous allez construire le système LFS en utilisant une distribution Linux déjà installée, telle que Debian, Mandrake, Red Hat ou SuSE. Ce système Linux existant (l'hôte) sera utilisé en tant que plateforme de démarrage, parce que vous aurez besoin d'outils comme un compilateur, un éditeur de liens et un shell pour construire le nouveau système. Normalement, tous les outils requis sont disponibles si vous avez sélectionné << développement >> comme option d'installation lorsque vous avez installé le système actuel.

Au chapitre 3, vous créerez d'abord une partition et un système de fichiers Linux, où vous compilerez et installerez votre nouveau système LFS. Puis dans le chapitre 4, vous téléchargerez tous les packages et correctifs requis pour construire un système LFS sur le nouveau système de fichiers.

chapitre 5 discutera de l'installation d'un certain nombre de packages formant la suite basique de développement (ou ensemble des outils) qui a été utilisée pour construire le système actuel au chapitre 6. Certains de ces packages sont nécessaires pour résoudre des dépendances circulaires. Par exemple, pour compiler un compilateur, vous avez besoin d'un compilateur.

La première chose à faire dans le chapitre 5 est de construire une première fois l'ensemble des outils, pour Binutils et GCC. Les programmes de ces packages seront liés statiquement pour être utilisés indépendamment du système hôte. La deuxième chose à faire est de construire Glibc, la bibliothèque C. Glibc sera coompilé avec les programmes de l'ensemble des outils que nous avons justement créé lors de la première passe. La troisième chose à faire est de construire une deuxième fois l'ensemble des outils. Cette fois, l'ensemble des outils sera lié dynamiquement avec la nouvelle construction de Glibc. Le reste des packages du chapitre 5 sont tous construits pendant cette deuxième passe et liés dynamiquement à la nouvelle Glibc indépendante de l'hôte. Ceci fait, le processus d'installation de LFS ne dépendra plus de la distribution hôte, avec l'exception du noyau en cours d'utilisation.

Vous pourriez vous dire << que cela semble être énormément de travail pour simplement s'éloigner de la distribution hôte >>. Une explication technique et complète est fournie au début du chapitre 5, incluant quelques notes sur les différences entre programmes liés statiquement et programmes liés dynamiquement.

Dans le chapitre 6, le vrai système LFS sera construit. Le programme chroot (change root) est utilisé pour entrer dans un environnement virtuel et pour lancer un nouveau shell dont le répertoire racine sera la partition LFS. Ceci est très similaire au redémarrage en indiquant au noyau que la partition racine est la partition LFS. La raison pour laquelle vous ne redémarrez pas réellement est que la création d'un système démarrable demande un travail supplémentaire inutile pour l'instant. Mais, l'avantage majeur de chroot est qu'il vous permet de continuer en utilisant l'hôte tant que LFS est en cours de construction. Alors que le logiciel est installé, vous pouvez simplement passer sur une autre console virtuelle ou bureau X et continuer à utiliser l'ordinateur comme d'habitude.

Pour finir l'installation, les scripts de démarrage sont configurés au chapitre 7, le noyau et le chargeur de démarrage dans le chapitre 8 et le chapitre 9 contient quelques pointeurs pour vous aider après avoir fini le livre. Enfin, vous être prêt à démarrer votre ordinateur avec votre nouveau système LFS.

En résumé, telle est la démarche à suivre. Des informations détaillées sur les différentes étapes vous seront fournies tout au long des chapitres au fur et à mesure de votre progression. Si tout n'est pas encore clair, ne vous inquiétez pas, cela ne saurait tarder.

Veuillez lire le chapitre 2 avec attention car il explique un certain nombre de points importants qu'il vous faut connaître avant d'en arriver au chapitre 5 et au−delà.

Conventions utilisées dans ce livre

Pour rendre les choses faciles à comprendre, il y a un certain nombre de conventions qui sont utilisées tout au long du livre. Voici quelques exemples :

```
./configure −−prefix=/usr
```

Cette façon de présenter montre les textes qui doivent être tapés exactement comme ils sont écrits, sauf si le texte autour dit le contraire. Cela est aussi utilisé dans les explications pour mettre en évidence les commandes auxquelles on fait référence.

```
install−info: unknown option `−−dir−file=/mnt/lfs/usr/info/dir'
```

Cette façon de présenter (texte de largeur fixe) montre les textes qui apparaissent à l'écran, très souvent comme résultat à la fin d'une commande. C'est aussi utilisé pour faire ressortir les noms de fichiers comme `/etc/ld.so.conf`.

Emphasis

Cette façon de présenter est utilisée dans ce livre pour différentes choses, principalement pour attirer l'attention sur des points importants mais aussi pour donner des exemples de ce qu'il faut taper.

http://www.linuxfromscratch.org/

Cette façon de présenter est utilisée pour les liens hypertextes internes à ce livre mais aussi externes comme les HOWTOs, les sites de téléchargement et les sites web.

```
cat > $LFS/etc/group << "EOF"
root:x:0:
bin:x:1:
......
EOF
```

Cette façon de présenter est utilisée principalement lorsqu'il y a création de fichiers de configuration. La première commande demande au système de créer le fichier $LFS/etc/group à partir des lignes qui suivent jusqu'à que la séquence EOF soit rencontrée. Par conséquent, l'ensemble de cette partie est généralement tapée comme elle est présentée.

Version du livre

Ceci est la version 5.0 du livre Linux From Scratch, daté du 5 novembre 2003. Si ce livre est âgé de plus de deux mois, une nouvelle, et certainement meilleure, version est probablement déjà disponible. Pour le savoir,

vérifiez sur un des miroirs listés sur http://www.linuxfromscratch.org/.

Journal des modifications (Changelog)

5.0 − 5 novembre 2003

- Mise à jour vers :
  - automake−1.7.6
  - bash−2.05b
  - binutils−2.14
  - e2fsprogs−1.34
  - file−4.04
  - findutils−4.1.20
  - gawk−3.1.3
  - gcc−3.3.1
  - gettext−0.12.1
  - glibc−2.3.2
  - glibc−2.3.2−sscanf−1.patch
  - grep−2.5.1
  - groff−1.19
  - gzip−1.3.5
  - less−381
  - lfs−bootscripts−1.12
  - libtool−1.5
  - linux−2.4.22
  - man−1.5m2
  - man−1.5m2−80cols.patch
  - man−1.5m2−manpath.patch
  - man−1.5m2−pager.patch
  - man−pages−1.60
  - modutils−2.4.25
  - procps−3.1.11
  - procps−3.1.11.patch
  - psmisc−21.3
  - sed−4.0.7
  - sysvinit−2.85
  - tar−1.13.25
  - texinfo−4.6
  - util−linux−2.12
  - vim−6.2
- Ajout de :
  - bash−2.05b−2.patch
  - bison−1.875−attribute.patch
  - coreutils−5.0
  - coreutils−5.0−uname.patch
  - coreutils−5.0−hostname−2.patch
  - dejagnu−1.4.3
  - expect−5.39.0

    - expect−5.39.0.patch
    - gawk−3.1.3.patch
    - gcc−2.95.3
    - gcc−2.95.3−2.patch
    - gcc−2.95.3−no−fixinc.patch
    - gcc−2.95.3−returntype−fix.patch
    - gcc−3.3.1−no_fixincludes−2.patch
    - gcc−3.3.1−specs−2.patch
    - gcc−3.3.1−suppress−libiberty.patch
    - grub−0.93
    - grub−0.93−gcc33−1.patch
    - inetutils−1.4.2
    - lfs−utils−0.3
    - ncurses−5.3−etip−2.patch
    - ncurses−5.3−vsscanf.patch
    - perl−5.8.0−libc−3.patch
    - shadow−4.0.3−newgroup−fix.patch
    - tcl−8.4.4
    - zlib−1.1.4−vsnprintf.patch
- Suppression de :

    - bin86−0.16.3
    - fileutils−4.1
    - fileutils−4.1.patch
    - findutils−4.1−segfault.patch
    - findutils−4.1.patch
    - glibc−2.3.1−libnss.patch
    - glibc−2.3.1−root−perl.patch
    - gzip−1.2.4b.patch
    - lilo−22.2
    - netkit−base−0.17
    - sh−utils−2.0
    - sh−utils−2.0.patch
    - sh−utils−2.0−hostname.patch
    - tar−1.13.patch
    - textutils−2.1
    - vim−6.1.patch
- 2 novembre 2003 [alex] : Annexe A − Mise en commentaire de toutes les lignes "Dernière vérification effectuée auprès de".
- 28 octobre 2003 [greg] : Renforcement des commandes sed dans les sections "Verrouiller Glibc" et "Réajuster l'ensemble des outils".
- 26 octobre 2003 [greg] : Chapitre 6 − Glibc : Ajout d'une commande pour créer /etc/ld.so.conf et correspondre ainsi au chapitre 5 sur Glibc. Ferme le bogue 700.
- 24 octobre 2003 [alex] : Annexe A − Passage des dépendances au nouveau format concis basé sur le mail de Tushar.
- 23 octobre 2003 [gerard] : Chapitre 9 − La fin : Modification du nom du fichier `/etc/lfs` en `/etc/lfs−release` pour être plus cohérent avec les autres distributions.
- 23 octobre 2003 [alex] : Changement de la plupart des références des chapitres en des balises "xref".
- 22 octobre, 2003 [alex] : Chapitre 6 − Gawk et Shadow : Ajustement du texte et ajout de quelques balises ailleurs.

- 22 octobre, 2003 [alex] : Chapitre 6 − Entrer dans l'environnement chroot : Suppression de la commande **`set +h`** car elle n'est pas nécessaire ici : elle est déplacée quelques sections après.
- 15 octobre, 2003 [greg] : Chapitre 9 − Refonte de la dernière commande strip. Déplacement des paragraphes sur la suppression de répertoires dans le chapitre 6.
- 14 octobre 2003 [greg] : Chapitre 8 − Rendre le système LFS démarrable : Plus de détails sur Grub et ajout d'un message d'avertissement.
- 14 octobre 2003 [alex] : Annexe A − Mise à jour du contenu de Perl et Procps.
- 14 octobre 2003 [alex] : Chapitre 4 and 5 − Ajout d'une suggestion pour utiliser $LFS/sources comme emplacement de travail et de stockage.
- 13 octobre 2003 [greg] : Chapitre 9 − Redémarrer le système : Refonte des commandes de démontage.
- 11 octobre 2003 [alex] : Mise à jour des valeurs d'espace disque requis et des SBU comme l'indiquait Bruce Dubbs.
- 11 octobre 2003 [alex] : Chapitre 5 − Notes techniques sur l'atelier d'outils : Ajout et modification de quelques balises.
- 9 octobre 2003 [gerard] : Mise à jour vers lfs−bootscripts−1.12.
- 9 octobre 2003 [greg] : Quelques travaux sur les balises internes pour corriger un problème d'espace blanc supplémentaire pour les pages HTML générées par tidy. Essentiellement, remplacement de toutes les occurences de <para><screen> par <screen> (et leur balise fermante).
- 9 octobre 2003 [alex] : Chapitre 6 − Réseau de base : Déplacement d'une moitié dans la section Lfs−Utils et de l'autre moitié vers Perl.
- 8 octobre 2003 [alex] : Chapitre 8 − Rendre le système LFS démarrable : Adaptation au style "screen" et modification de quelques paragraphes.
- 8 octobre 2003 [alex] : Suppression d'une série d'entités inutilisées.
- 7 octobre 2003 [jeremy] : Ajout de notes pour les tests d'édition de liens dans les chapitres 5 et 6 indiquant qu'une sortie vierge est une mauvaise chose.
- 7 octobre 2003 [alex] : Modification des entités de correctifs pour contenir le nom complet au lieu du seul numéro de version.
- 7 octobre 2003 [jeremy] : Chapitre 1 − Ajout d'une note concernant #LFS−support sur IRC.
- 7 octobre 2003 [greg] : Preface − Ajout d'une note concernant l'astuce LFS contenant des informations essentielles avant cette lecture. Ferme le bogue 585.
- 6 octobre 2003 [alex] : Modification du style des sous−sections de contenus des chapitres 5 et 6 ainsi que de l'annexe A.
- 6 octobre 2003 [greg] : Simplification des commandes sed dans les sections "Verrouiller Glibc" et "Réajustement de la suite d'outils". Modification de la section "Comment ceci va se passer".
- 5 octobre 2003 [greg] : Chapitre 5 − Ajout d'une section "Notes techniques sur l'atelier des outils". Intégration et modification de l'ancienne section "Pourquoi nous utilisons des liens statiques". Ferme le bogue 658.
- 4 octobre 2003 [alex] : Quelques modifications mineures et ajout de balises un peu partout.
- 4 octobre 2003 [greg] : Chapitre 5 − Binutils Passe 1 : Ajout d'un LDFLAGS supplémentaire pour nous assurer de la reconstruction statique de ld.
- 2 octobre 2003 [greg] : Chapitre 6 − Remise en place de INSTALL=/tools/bin/install pour la commande d'ajustement de l'éditeur de liens dû à des problèmes sur les hôtes où un lien symbolique ginstall existe. Ceci rend les liens symboliques "install" redondants, donc suppression de ces derniers.
- 2 octobre 2003 [greg] : Chapitre 6 − Shadow : Activation des mots de passe MD5. Ferme le bogue 600.
- 27 septembre 2003 [greg] : Chapitre 5 − Expect : Amélioration de l'installation en faisant que les script redondants ne soient pas installés. Chapitre 6 − Création des liens symboliques essentiels. Chapitre 6 − man : Suppression de PATH, fin du bogue 574.
- 27 septembre 2003 [greg] : Ajout des éléments Tcl, Expect et DejaGnu à l'annexe A. Fin du bogue 661.

- 26 septembre 2003 [jeremy] : Ajout d'un nouveau correctif pour les problèmes de devpts
- 24 septembre 2003 [alex] : Annexe A − Modification du style des courtes descriptions et du contenu de la plupart d'entre eux.
- 24 septembre 2003 [greg] : Différentes modifications concernant le bogue 675.
- 22 septembre 2003 [greg] : Chapitre 8 − Créer le fichier /etc/fstab : Monter devpts est fait par défaut.
- 22 septembre 2003 [jeremy] : Ajout d'un correctif pour Net−tools corrigeant la compilation de mii−tool
- 22 septembre 2003 [jwrober] : Chapitre 5 − Mise à jour de la page "Pourquoi utiliser une édition de lien statique ?" pour représenter plus fidèlement les différences entres des binaires liés statiquement et des binaires liés dynamiquement. Merci à Ian Molton pour ces précisions. Corrige le bogue 602.
- 22 septembre 2003 [jeremy] : Suppression de la commande make de DejaGnu, car elle ne réalise rien.
- 22 septembre 2003 [jeremy] : Suppression du −k du make check de Tcl, car on ne s'attend plus à avoir encore des erreurs.
- 22 septembre 2003 [jeremy] : Modification de la référence vers l'astuce de man en un pointeur vers BLFS
- 22 septembre 2003 [jeremy] : Ajout d'une note pour se rappeler de monter devpts si vous sortez puis réentrez dans chroot
- 22 septembre 2003 [jeremy] : Suppression du make check pour patch et diffutils, car ces tests ne réalisent rien.
- 22 septembre 2003 [greg] : Chapitre 5 − Initialiser l'environnement : Ajout de "unset CC CXX CPP LD_LIBRARY_PATH LD_PRELOAD" dans .bash_profile pour stopper les accidents lors de la construction.
- 20 septembre 2003 [greg] : Chapitre 5 − GCC Passe 2 : Mise à jour vers GCC−3.3.1−specs−2.patch. Ncurses : ajout de −−enable−overwrite et description.
- 19 septembre 2003 [jeremy] : Correction des balises bash par une utilisation plus propre de l'option +h de bash.
- 19 septembre 2003 [jwrober] : Différentes mises à jour pour la page des remerciements.
- 18 septembre 2003 [jeremy] : Chapitre 5 − GCC Passe 2 − ajout de quelques commentaires supplémentaires concernant les trois archives tar à déballer.
- 17 septembre 2003 [greg] : Chapitre 6 − GCC−2.95.3: Ajout de notes rationnelles.
- 17 septembre 2003 [jwrober] : Mise à jour de la page des remerciements pour correspondre au site web.
- 17 septembre 2003 [jeremy] : Mise à jour de File en 4.04.
- 17 septembre 2003 [jeremy] : Chapitre 6 − Modification de deux des occurences de exec bash −−login pour inclure la directive +h.
- 17 septembre 2003 [greg] : Chapitres 5 et 6 − Verrouillage de Glibc et réajustement de l'ensemble des outils : Faire "make −C ld install" au lieu de "make −C ld install−data−local" pour installer un nouvel éditeur de liens plutôt que simplement les nouveaux ldscripts.
- 17 septembre 2003 [alex] : Normalisation de l'orthographe de 'Tcl' et 'DejaGnu', suivant leur propre documentation.
- 17 septembre 2003 [alex] : Correction au niveau des dépendances.
- 16 septembre 2003 [greg] : Chapitres 5 et 6 − Verrouillage de Glibc et Réajustement de l'ensemble des outils : Ajout de vérifications.
- 16 septembre 2003 [greg] : Chapitres 5 et 6 − Binutils, GCC, et Glibc : Ajout de notes sur les suites de test.
- 15 septembre 2003 [greg] : Chapitre 6 − Révision de la commande chroot : Suppression de +h, inutile.
- 15 septembre 2003 [alex] : Correction de plusieurs erreurs et inconsistances.
- 14 septembre 2003 [alex] : Correction de quelques erreurs et ajout de balises, Suppression de la suppression des programmes dans la section "Stripping" du chapitre 5.
- 14 septembre 2003 [greg] : Chapitre 6 − Créer les liens symboliques essentiels : Ajout d'un lien symbolique /usr/lib/libgcc_s.so.1 pour permettre à gcc abi_check de se lancer. La future NPTL en a

aussi besoin.

- 13 septembre 2003 [jwrober] : Ajout de texte de l'astuce PLFS sur la page du chapitre 6 pour créer passwd et group : bogue 596.
- 13 septembre 2003 [jwrober] : Mise à jour de la page "Comment les choses se dérouleront−elles" pour inclure plus de texte de l'astuce PLFS.
- 13 septembre 2003 [jwrober] : Préface − Assemblage des pages "Qui devrait lire" et "Qui ne devrait pas lire" en une seule page.
- 13 septembre 2003 [greg] : Chapitre 2 − Ajout d'une nouvelle section concernant les suites de tests.
- 12 septembre 2003 [jeremy] : Chapitre 5 − Ncurses : Ajout de la description de l'option configure −−without−ada.
- 12 septembre 2003 [jeremy] : Chapitre 5 − Gawk : Ajout de la suite de tests.
- 12 septembre 2003 [jeremy] : Chapitre 5 − Grep : Ajout des descriptions des options configure grâce à Anderson Lizardo.
- 12 septembre 2003 [gerard] : Suppression de la création du répertoire /usr/lib/locale − il est créé lors du chapitre 6 − Glibc.
- 11 septembre 2003 [jwrober] : Chapitre 5 − Correction du chapitre 5 GCC Passe 2 en ce qui concerne le texte du correctif specs pour être plus vague, mais en fait plus aiguisé − fourni par Anderson Lizardo.
- 11 septembre 2003 [jwrober] : Chapitre 5 − Correction grammaticale dans le chapitre 5 pour les instructions d'installation de Tcl, fournis par Anderson Lizardo.
- 11 septembre 2003 [jwrober] : Chapitre 5 − Quelques petites modifications sur le texte dans la page "verrouillage de Glibc" au chapitre 5 pour /lib/ld.so.1 fourni pas Anderson Lizardo.
- 11 septembre 2003 [jeremy] : Ajout de la configuration du chargeur de démarrage dans le chapitre 8, après l'ajout de Grub dans le livre.
- 11 septembre 2003 [gerard] : Suppression de Bin86 et LILO en les remplaçant par Grub.
- 11 septembre 2003 [jeremy] : Passage des tests non relatifs à l'ensemble des outils en action en option. Ajout d'une note pour utiliser le wiki pour les tests ayant échoué.
- 11 septembre 2003 [jeremy] : Ajout du correctif pour Bison, suite à un rétro−portage de CVS, pour corriger les problèmes de compilation de pwlib.
- 11 septembre 2003 [jeremy] : Ajout du correctif de Greg à gcc pour supprimer l'installation de libiberty, et permettre à binutils de conserver son libiberty.
- 11 septembre 2003 [jeremy] : Ajout de balises d'attention autour du rappel de ne pas supprimer les répertoires des sources et de construction de binutils dans le chapitre 5.
- 11 septembre 2003 [jeremy] : Ajout du correctif pour le nouveau perl−libc−3 de Anderson Lizardo.
- 9 septembre 2003 [jwrober] : Correction du lien de téléchargement du package findutils sur la page des packages fermant ainsi le bogue 578.
- 9 septembre 2003 [jeremy] : Chapitre 6 − GCC 2.95.3 : Suppression de la compilation de C++, ajout du correctif return−type de Zac.
- 9 septembre 2003 [jeremy] : Chapitre 6 − Coreutils : Ajout de coreutils−5.0−hostname−2.patch, qui supprime la construction du binaire hostname, et supprime aussi sa vérification.
- 9 septembre 2003 [jeremy] : Ajout de quelques notes concernant les tests échoués dans Glibc et DejaGnu.
- 9 septembre 2003 [jeremy] : Glibc − Ajout de commandes à la fois au chapitre 5 et 6 pour inclure le minimum de locales nécessaire aux vérification.
- 9 septembre 2003 [jeremy] : Chapitre 6 − Suppression du point sur zlib pour CFLAGS pour une note pour ajouter −fPIC.
- 8 septembre 2003 [matt] : Chapitre 5 − Correction de la commande rm supprimant une documentation non nécessaire dans /tools/share.
- 6 septembre 2003 [matt] : Chapitre 6 − Suppression d'une référence vers le répertoire "statique" dans l'introduction.

- 6 septembre 2003 [jeremy] : Chapitre 4 − Mise à jour de l'emplacement de téléchargement pour certains packages.
- 5 septembre 2003 [jeremy] : Chapitre 5 − GCC Passe 2 : Corrrection de l'explication sur l'erreur de make check.
- 5 septembre 2003 [jeremy] : Chapitre 6 − Makedev : Modification de la création des périphériques par défaut avec generic−nopty, parce que nous utilisons maintenant devpts par défaut.
- 5 septembre 2003 [jeremy] : Chapitre 6 − GCC : Correction de la phrase pour reflecter la suppression du lien symbolique /usr/lib/cpp.
- 5 septembre 2003 [jeremy] : Modification du correctif libc de perl en −2, renommage de l'ancienne structure /stage1 en /tools.
- 5 septembre 2003 [matt] : Chapitre 6 − Mise à jour du correctif specs de gcc et mise à jour vers man−1.5m2
- 4 septembre 2003 [jeremy] : Chapitre 6 − Création des répertoires : Suppression de la création de /usr/tmp − Fin du bogue 176.
- 4 septembre 2003 [jeremy] : Chapitre 6 − Monter Proc : Ajout du montage du système de fichiers devpts avec chroot ici. Fin du bogue 533.
- 4 septembre 2003 [jeremy] : Chapitre 6 − Monter Proc : Ajout d'un avertissement à la fin concernant la vérification que proc est bien toujours monté si vous stoppez puis relancez le processus lfs.
- 4 septembre 2003 [jeremy] : Chapitre 6 − Gzip : Modification du texte pour mieux expliquer la raison derrière la commande sed utilisée lors de l'installation de gzip. Fin du bogue 551.
- 4 septembre 2003 [jeremy] : Chapitre 4 − Téléchargement des correctifs : Ajout d'une note concernant le projet des correctifs de Tushar et un lien vers la page d'accueil des correctifs.
- 3 septembre 2003 [matt] : Correction du problème avec util−linux pour la non utilisation des entêtes et bibliothèques installées dans /stage1.
- 3 septembre 2003 [matt] : Suppression de l'instruction "rm /bin/pwd" provenant de l'installation des entêtes du noyau lors du chapitre 6 car le lien est toujours nécessaire pour l'installation de Glibc.
- 2 septembre 2003 [alex] : Ajustement de tous les SBU à partir des valeurs postées par Jeremy − s'il ne m'a pas battu pour le faire.
- 2 septembre 2003 [alex] : Assemblage de plusieurs fichiers de structure du livre.
- 2 septembre 2003 [alex] : Classement en ordre alphabêtique des listes de téléchargements, ajout d'une note pour les instructions sur Tcl.
- 2 septembre 2003 [alex] : Réécriture des sections sur l'organisation, $LFS et SBU.
- 1er septembre 2003 [jeremy] : Chapitre 6 − Groff : Ajout d'une note sur le choix de A4 ou letter pour la variable PAGE.
- 1er septembre 2003 [jeremy] : Ajout dans shadow du correctif newgrp provenant de Greg Schafer
- 31 août 2003 [jeremy] : − Chapitre 6 − Inetutils : Ajout des options −−disable−whois et −−disable−servers.
- 31 août 2003 [jeremy] : − Ajout des nouvelles instructions de Greg pour GCC 3.3.1 avec le processus pour fixincludes. Ainsi que l'ajout de phrases dans les pages "verrouillage" et "deuxième passe de GCC" sur le processus de fixincludes.
- 31 août 2003 [alex] : Remise en forme de certains paragraphes, ajout de balises manquantes markup et réarrangement du journal des modifications.
- 31 août 2003 [alex] : Mise entre parenthèses des lignes de "Dernière vérification". Quelques autres petites retouches.
- 30 août 2003 [jeremy] : Mise à jour du correctif fix−includes pour GCC 3.3.1
- 29 août 2003 [alex] : Suppression des fichiers obsolètes de Netkit−base, Fileutils, Sh−utils et Textutils.
- 29 août 2003 [alex] : Ajout de quelques balises manquantes, modification de quelques /static en /stage1.
- 29 août 2003 [alex] : Chapitre 06 − Ajout de toutes les lignes de texte manquantes avant les make check, et refonte de certaines lignes.

- 29 août 2003 [jeremy] : − Glibc − Mise à jour des instructions pour le correctif sscanf.
- 29 août 2003 [jeremy] : − Mise à jour de GCC à la version 3.3.1, avec les correctifs basés sur la mini astuce de Zack pour GCC 3.3 et les correctifs provenant de ces documents.
- 28 août 2003 [matt] : − Mise à jour de certains packages... linux−2.4.22, man−pages−1.60, expect−5.39.0, findutils−4.1.20 et tcl−8.4.4.
- 28 août 2003 [jeremy] : − Nouveau fichier bash−2.05b−2.patch pour inclure les sept correctifs de ftp.gnu.org
- 28 août 2003 [alex] : Chapitre 06 − Réajuster l'ensemble des outils : Ajout d'un antislash oublié.
- 28 août 2003 [alex] : Correction de quelques erreurs et ajout de balises manquantes.
- 28 août 2003 [alex] : Chapitre 06 − Binutils et GCC : Intégration de texte provenant de l'astuce pure−lfs.
- 27 août 2003 [alex] : Chapitre 06 − Glibc : Intégration de texte provenant de l'astuce pure−lfs.
- 27 août 2003 [jeremy] : − Chapitre 06 − Inetutils : Ajout de −−sysconfdir=/etc −−localstatedir=/var et déplacement du binaire ping de /usr/bin dans /bin.
- 26 août 2003 [alex] : Chapitre 06 & 08 − Déplacement de l'installation des pages man du noyau du chapitre 6 au chapitre 8.
- 26 août 2003 [jeremy] : − Chapitre 07 − Créer /etc/hosts : Modification de www.mydomain.org en <value of HOSTNAME>.mydomain.org.
- 26 août 2003 [jeremy] : − Chapitre 04 − Monter la partition LFS : Ajout d'un texte concernant le montage avec des droits trop restrictifs.
- 26 août 2003 [jeremy] : − Chapitre 06 − Créer les répertoires : Ajout de la création du répertoire /dev/shm.
- 26 août 2003 [jeremy] : − Chapitre 08 − Créer fstab : Ajout du système de fichiers tmpfs à /dev/shm.
- 26 août 2003 [jeremy] : − Chapitre 08 − Installation du noyau : Ajout d'un rappel pour compiler le support de tmpfs dans le noyau.
- 25 août 2003 [alex] : Chapitre 06 − Réécriture du texte d'installation de Shadow et Util−Linux en corrigeant certaines erreurs de typographie.
- 25 août 2003 [alex] : Chapitre 05 & 06 − Fait en sorte que "Verrouillage" et "Réajustement" se ressemblent.
- 24 août 2003 [alex] : Chapitre 04 − Assemblage des trop nombreux petits fichiers en un seul. Les packages et les correctifs ont chacun une page séparée.
- 17 août 2003 [alex] : Chapitre 05 − De Bash à Perl : mise en place de texte entre les commandes. Ajout d'une section sur la suppression des symboles inutiles pour diminuer la taille des outils.
- 16 août 2003 [alex] : Chapitre 05 − De Make à Texinfo : mise en place de texte entre les commandes.
- 11 août 2003 [alex] : Chapitre 05 − De la première passe pour Binutils à Findutils : plusieurs petits ajustements de texte. Pour les deuxièmes passes sans donner le contenu et les dépendances.
- 11 août 2003 [alex] : Chapitre 04 − Liste pour des archives séparées du coeur de GCC, de g++ et des suites de test.
- 11 août 2003 [alex] : Chapitre 04 − Suppression de la mention d'un script wget.
- 9 août 2003 [alex] : Chapitre 05 − Binutils Passe 2 et GCC Passe 2 : intégration de texte provenant de l'astuce pure−lfs.
- 8 août 2003 [alex] : Chapitre 05 − Tcl, Expect et DejaGnu : ajout de texte.
- 6 août 2003 [gerard] : Application du correctif d'Alex Groenewoud, ajoutant pour l'annexe B une liste de tous les programmes et de toutes les bibliothèques installés avec la référence à leur page d'installation.
- 30 juillet 2003 [gerard] : Chapitre 06 − Vim : Modification des emplacements des fichiers `vimrc` et `gvimrc`.
- 30 juillet 2003 [gerard] : Chapitre 05 − Binutils Pass2 : Suppression du correctif lib, il n'est plus nécessaire avec la mise à jour binutils−2.14 .
- 30 juillet 2003 [gerard] : Chapitre 05 − Binutils Pass1 : Ajout de **`make configure-host`**.

- 30 juillet 2003 [gerard] : Mise à jour vers binutils−2.14, linux−2.4.21, expect−5.38.4, gawk−3.1.3, texinfo−4.6, util−linux−2.12, man−pages−1.58, lfs−utils−0.3, vim−6.2, gettext−0.12.1, automake−1.7.6, file−4.03, e2fsprogs−1.34, procps−3.1.11, psmisc−21.3
- 3 juin 2003 [gerard] : Chapitre 06 − Gawk : suppression de la suppression de `/bin/awk`. Ce lien symbolique n'est plus créé.
- 21 mai 2003 [gerard] : Chapitre 06 − GCC−2.95.3 : Ajout de /opt/gcc−2.95.3/lib au fichier /etc/ld.so.conf de façon à ce que les bibliothèques soient détectées au démarrage.
- 21 mai 2003 [gerard] : Chapitre 05 − Gzip : Simplification des commandes.
- 21 mai 2003 [gerard] : Chapitre 05 − Bzip2 : Simplification des commandes.
- 21 mai 2003 [gerard] : Chapitre 06 − Shadow : Ajout de la commande **`grpconv`** pour compléter l'activation de tous les mots de passe shadow.
- 21 mai 2003 [winkie] : Chapitre 06 − Créer les fichiers : Toutes les commandes **`ln`** peuvent être remplacées par quelques commandes ln longues.
- 21 mai 2003 [winkie] : Chapitre 05 − Installer Glibc : Créer un fichier ld.so.conf avant de construire Glibc, pour prévenir une erreur (non dangereuse).
- 21 mai 2003 [winkie] : Chapitre 06 − Installer Glibc : Ne pas s'embêter à faire le 'exec /stage1/bin/bash', cela ne fait plus rien maintenant que nous utilisons PLFS.
- 21 mai 2003 [winkie] : Chapitres 05 & 06 − Installer Coreutils : Tester seulement la partie non−root du chapitre 5, mais tout tester au chapitre 6.
- 21 mai 2003 [winkie] : Chapitre 05 − Installer Expect : Ne passe rien de plus que −−prefix=/stage1. Rien de plus n'est nécessaire.
- 16 mai 2003 [gerard] : Chapitre 06: Net−tools : Changement de **`make install`** en **`make update`**.
- 15 mai 2003 [timothy] : Chapitre 05 − Installer Patch : Ajout de **`CPPFLAGS=−D_GNU_SOURCE`** avant le **`./configure`** pour corriger un problème posé par le correctif pour les PPC.
- 13 mai 2003 [gerard] : Chapitre 06 : Lorsque nous exécutons **`exec /path/to/bash −−login`**, nous devons aussi lancer **`set +h`** pour garder l'option de hachage désactivée. Correction du bogue #531
- 13 mai 2003 [gerard] : Chapitre 06 − Réseau de base : Modification des simples quotes en double quotes pour la commande echo. Sans cela, $(hostname) ne pourra pas être évaluée ce qui empêchera la seule raison d'être de cette commande − faire que la vérification de hostname par Perl fonctionne.
- 13 mai 2003 [winkie] : Suppression de toutes les occurrences &&. Mise à jour de la syntaxe des bogues. Ajout de "make check/test" là où c'était nécessaire dans le chapitre 6.
- 13 mai 2003 [winkie] : Chapitre 6 − Application du correctif de la partie "Changer le propriétaire" pour améliorer le texte. Fin du bogue #511.
- 13 mai 2003 [winkie] : Chapitre 6 − Application du correctif de la partie "Configurer les composants du système" pour améliorer le texte. Fin du bogue #510.
- 13 mai 2003 [gerard] : Chapitre 06 : Suppression de TCL, Expect et DejaGNU. Aucune utilisation de ceux−ci une fois que GCC est installé dans le chapitre 6. Les versions dans /stage1/bin font leur travail correctement.
- 13 mai 2003 [winkie] : Chapitre 06 − Installer Shadow : Création du fichier /usr/bin/passwd (avec touch) avant l'installation. Si on ne le fait pas, Shadow pensera qu'il se trouve dans /bin/passwd.
- 13 mai 2003 [winkie] : Chapitre 06 − Installer Procps : Suppression du lien symbolique /lib/libproc.so. Aucun package en dehors de Procps lui−même n'utilise cette bibliothèque, et en fait, personne ne le devrait.
- 13 mai 2003 [winkie] : Chapitre 06 − Installer Net−tools : Lancement de "make config" avant de lancer make. Corrige les bogues #462 et #497.
- 13 mai 2003 [gerard] : Chapitre 06 − Ncurses : Ajout du correctif vsscanf.
- 12 mai 2003 [gerard] : Chapitre 05 − Gzip : Suppression de **`make check`**. Il ne faisait rien.
- 12 mai 2003 [winkie] : Chapitre 05 − Installer Texinfo : N'installez pas les données texmf. Il n'est pas utilisé.

- 12 mai 2003 [winkie] : Chapitre 05 & 06 − Installer Ncurses : Dans le chapitre 6, la création du lien symbolique a été mise à jour pour inclure libcurses.*, et les droits de libncurses++.a sont maintenant modifiés en 644. Le chapitre 5 n'a besoin d'aucun libcurses.*, donc ils sont supprimés.
- 12 mai 2003 [gerard] : Chapitre 06 − Réseau de base : Ajout de $(hostname) dans /etc/hosts, sinon le test d'hostname par Perl echouera.
- 12 mai 2003 [gerard] : Chapitre 06 − Installer GCC : N'essayer pas de supprimer /usr/include/libiberty.h. Il n'est pas installé au début.
- 12 mai 2003 [winkie] : Mise à jour vers findutils−4.1.7, gzip−1.3.5 et tar−1.13.25.
- 12 mai 2003 [winkie] : Chapitre 05 − Installer Perl : Ajout des commandes supplémentaires pour construire certains modules dans Perl. Ceci permet d'accomoder le "make check" de Coreutils. Corrige partiellement le bogue #528.
- 12 mai 2003 [winkie] : Chapitre 05 − Installer Gzip : Aucun package dans le chapitre 6 ne vérifie ou n'utilise la commande uncompress, du coup nous ne devrions pas la créer.
- 12 mai 2003 [winkie] : Chapitre 05 − Installer Bzip2 : Exécuter "make" implique "make check", donc il n'existe pas de raison pour que nous le lancions manuellement.
- 12 mai 2003 [winkie] : Chapitre 05 − Installer Lfs−Utils : Supprimé. Le seul package qui vérifie l'existence de mktemp avant d'être installé est GCC pour la commande gccbug.
- 11 mai 2003 [gerard] : Chapitre 06 − GCC−2.95.3 : Ajout de −−enable−threads=posix pour compléter l'ajout de C++.
- 11 mai 2003 [gerard] : Chapitre 06 − GCC−2.95.3 : Ajout de −−enable−languages=c,c++ pour corriger ce bogue de cette version de gcc, concernant −Wreturn−type.

  Corrige le bogue #525.
- 11 mai 2003 [gerard] : Chapitre 05 − Bash : Suppression de l'option de configure −−without−bash−malloc.
- 11 mai 2003 [gerard] : Mise à jour vers gcc−3.2.3−specs−4.patch.
- 11 mai 2003 (Bug #359 & #515) [winkie] : Chapitre 06 − Configuration basique du réseau : Ajout de cette section. Création d'un fichier /etc/hosts basique, et ajout des fichiers /etc/services et /etc/protocols en provenance de l'IANA.
- 11 mai 2003 [winkie] : Mise à jour vers lfs−utils−0.2.2. Ceci ajoute deux fichiers nécessaires pour une configuration correcte du réseau.
- 11 mai 2003 (Bug #490) [winkie] : Suppression de Netkit−base 0.17. Ajout de Inetutils 1.4.2.
- 11 mai 2003 (Bug #493) [winkie] : Ajout de lfs−utils−0.2.1.
- 11 mai 2003 [winkie] : Chapitre 06 − Installer Ncurses : Correction des liens symboliques pour qu'elles suivent les autres liens symboliques de bibliothèques. Rien d'étrange ici.
- 11 mai 2003 [winkie] : Chapitre 06 − Installer Procps : Suppression de XSCPT="" et de son paragraphe correspondant. Ce n'est plus utile.
- 11 mai 2003 [winkie] : Chapitre 06 − Installer Ncurses : Ajout de −−without−debug au script configure. Il semble avoir été perdu à un moment.
- 11 mai 2003 [timothy] : Chapitre 5 & 6 − Installer Bzip2, Installer Zlib: Modification des commandes de construction suivant le bogue #524.
- 11 mai 2003 [winkie] : Chapitre 06 − Installer Glibc : Installation des pages man de linuxthreads. Ceci a dû se perdre quelque part.
- 11 mai 2003 [winkie] : Chapitre 06 − Installer Grep : Ajout de −−with−included−regex pour empêcher Grep d'utiliser les regex boguées de Glibc.
- 11 mai 2003 [winkie] : Chapitre 06 − Installer Coreutils : Corrige quelques fonctionnalités de la commande uname.
- 11 mai 2003 [winkie] : Chapitre 06 − Installer Net−tools : Fait simplement un "make install" standard au lieu d'un "make update". Cela fonctionne bien maintenant.
- 11 mai 2003 [winkie] : Chapitre 06 − Installer GCC : Après installation, supprimez /usr/include/libiberty.h. Il n'est pas utile en dehors du répertoire de construction de GCC.

- 11 mai 2003 [winkie] : Mise à jour vers Bash 2.05b ainsi que son correctif.
- 11 mai 2003 [winkie] : Chapitre 06 − Installer Zlib : Application d'un correctif pour le dépassement de tampon de la fonction gzprintf().
- 11 mai 2003 [winkie] : Chapitre 06 − Configurer les composants du système : Déplacement de la création de btmp, wtmp, lastlog et utmp juste avant Shadow, de façon à ce qu'ils soient détectés dans leur bon emplacement.
- 10 mai 2003 [winkie] : Chapitre 06 − Installer Automake : Lancer "make" avant d'installer. Ceci est nécessaire maintenant avec les nouvelles versions d'Automake.
- 10 mai 2003 [winkie] : Chapitre 06 − Installer Vim : Suppression du correctif. Il n'est plus requis depuis GCC 3.2.1.
- 10 mai 2003 [winkie] : Chapitre 06 − Créer le fichier mtab : Supprimé. Monter /proc a comme effet de bord de créer /etc/mtab pour nous.
- 10 mai 2003 [winkie] : Chapitre 06 − Installer Make : Suppression de la modification de /usr/bin/make file. Il n'est plus installé par erreur avec des droits et une appartenance étranges.
- 10 mai 2003 [winkie] : Chapitre 06 − Installer Glibc : /etc/localtime est maintenant un fichier au lieu d'un lien symbolique. La méthode du lien ne fonctionne pas sur les systèmes où /usr se trouve dans une partition séparée.
- 10 mai 2003 [winkie] : Chapitre 06 − Installer E2fsprogs : Suppression des commandes install−info pour e2fsprogs. La cible "make install" le gère pour nous.
- 10 mai 2003 [gerard] : Suppression de toutes les variables CFLAGS et LDFLAGS où elles ne sont pas essentielles (donc, n'incluant pas les version statiques de binutils, gcc et la compilation de zlib avec −fPIC).
- 10 mai 2003 [gerard] : Chapitre 05 − Binutils (passe 1, passe 2, verrouillage de Glibc et ajustement de l'ensemble des outils) : Déplacement de l'ensemble des outils dans /stage1 (mais nous utilisons tooldir=/usr au chapitre 6).
- 10 mai 2003 [gerard] : Chapitre 05 − Entêtes du noyau : Suppresion de **`cp -H`** parce que certaines distributions ne connaissent pas l'option **`-H`**.
- 10 mai 2003 [gerard] : Nouveau gcc−3.2.3−specs−3.patch.
- 10 mai 2003 [gerard] : Chapitre 06 − Ajustement de l'ensemble d'outils : Rendu plus indépendant par rapport à l'architecture.
- 10 mai 2003 [gerard] : Chapitre 05 − Verrouillage dans Glibc : Rendu plus indépendant par rapport à l'architecture.
- 7 mai 2003 [gerard] : Suppression des correctifs "No Debug" de GCC. Nous ne supposons plus que les packages gcc−core et gcc−g++ ont été téléchargés, donc ajout des options −−enable−languages appropriées.
- 7 mai 2003 [gerard] : Suppression du Chapitre 6 − Glibc−Pass2. Il n'est plus nécessaire depuis l'intégration de pure−lfs.
- 7 mai 2003 [gerard] : Retour une nouvelle fois à la version flex−2.5.4a again. Les nouvelles versions ne fonctionnent tout simplement pas correctement.
- 5 mai 2003 [gerard] : Suppression de l'installation de zlib lors du chapitre 5 (son ajout était une erreur).
- 5 mai 2003 [gerard] : Différents corrections dues à l'intégration de pure−lfs.
- 2 mai 2003 [gerard] : Mise à jour vers : automake−1.7.4, e2fsprogs−1.33, file−4.02, flex−2.5.31, gawk−3.1.2, gcc−3.2.3, glibc−2.3.2, grep−2.5.1, groff−1.19, less−381, libtool−1.5, man−1.5l, man−pages−1.56, modutils−2.4.25, procps−3.1.8, sed−4.0.7, sysvinit−2.85, texinfo−4.5, util−linux−2.11z
- 2 mai 2003 [gerard] : Suppression de fileutils−4.1, sh−utils−2.0, textutils−2.1 (tous remplacé par coreutils−5.0).
- 2 mai 2003 [gerard] : Ajout de binutils−2.13.2−libc.patch, coreutils−5.0, dejagnu−1.4.3, expect−5.38, gawk−3.1.2, gcc−2.95.3, tcl−8.4.2

- 2 mai 2003 [gerard] : − Intégration de la nouvelle méthode d'installation suivant l'astuce Pure LFS écrit par Greg Schafer et Ryan Oliver.

Sortie de la version 4.1 le 28 avril 2003.

Ressources

FAQ

Si, lors de la construction de votre système LFS, vous rencontrez des erreurs, si vous vous posez des questions ou si vous pensez avoir trouvé une erreur dans le livre, merci de consulter en premier lieu la FAQ (Foire Aux Questions) sur http://www.linuxfromscratch.org/faq/.

IRC

Plusieurs membres de la communauté LFS offrent leur aide sur le serveur IRC de la communauté. Avant d'utiliser ce type d'aide, nous vous demandons d'avoir au moins vérifié la FAQ LFS et les archives des listes de diffusion pour une réponse à votre question. Vous trouverez le serveur IRC sur *irc.linuxfromscratch.org* port 6667. Le canal d'aide est nommé #LFS−support.

Listes de diffusion

Le serveur linuxfromscratch.org est l'hôte d'un certain nombre de listes de diffusion utilisées pour le développement du projet LFS. Ces listes incluent, entre autres, les listes principales de développement et de support.

Pour plus d'informations concernant les listes disponibles, la façon d'y souscrire, l'emplacement des archives, jetez un oeil sur http://www.linuxfromscratch.org/mail.html

Serveur News

Toutes les listes de diffusion hébergées par linuxfromscratch.org sont aussi accessibles via le serveur NNTP. Tous les messages postés sur la liste de diffusion sont copiés vers le newsgroup correspondant, et vice versa.

Le serveur news peut être accédé par *news.linuxfromscratch.org*.

Sites miroirs

Le projet LFS dispose de plusieurs miroirs configurés dans le monde pour vous permettre plus facilement et d'une façon plus confortable de vous y connecter et d'y télécharger les packages requis. Visitez le site http://www.linuxfromscratch.org pour la liste des miroirs actuels.

Contacts et informations

Envoyez de préférence tous vos emails vers l'une des listes de diffusion. Voir la section intitulée *Listes de diffusion* pour plus d'informations sur les listes de diffusion disponibles.

Mais si vous avez besoin de joindre Gerard Beekmans personnellement, envoyez un email à gerard@linuxfromscratch.org.

Remerciements

Nous souhaitons remercier les personnes et organisations suivantes pour leurs contributions au projet Linux From Scratch.

Membres actuels de l'équipe du projet

- Gerard Beekmans <gerard@linuxfromscratch.org> −− Initiateur du projet Linux−From−Scratch, Organisation du projet LFS.
- Matthew Burgess <matthew@linuxfromscratch.org> −− Mainteneur des packages généraux de LFS, éditeur du livre LFS.
- Craig Colton <meerkats@bellsouth.net> −− Créateur des logos des projets LFS, ALFS, BLFS et Hints.
- Jeroen Coumans <jeroen@linuxfromscratch.org> −− développeur du site web et mainteneur de la FAQ.
- Bruce Dubbs <bdubbs@linuxfromscratch.org> −− Chef de l'équipe d'assurance qualité, éditeur du livre BLFS.
- Alex Groenewoud <alex@linuxfromscratch.org> −− Editeur du livre LFS.
- Mark Hymers <markh@linuxfromscratch.org> −− Mainteneur du CVS, créateur du livre BLFS, ancien éditeur du livre LFS.
- James Iwanek <iwanek@linuxfromscratch.org> −− Membre de l'équipe de l'administration système.
- Nicholas Leippe <nicholas@linuxfromscratch.org> −− Mainteneur du Wiki.
- Anderson Lizardo <lizardo@linuxfromscratch.org> −− Créateur et mainteneur des scripts du moteur du site web.
- Bill Maltby <bill@linuxfromscratch.org> −− Organisation du projet LFS.
- Scot Mc Pherson <scot@linuxfromscratch.org> −− Mainteneur de la passerelle NNTP de LFS.
- Ryan Oliver <ryan@linuxfromscratch.org> −− Responsable de l'équipe de test, co−créateur de PLFS.
- James Robertson <jwrober@linuxfromscratch.org> −− Mainteneur de Bugzilla, développeur du Wiki, éditeur du livre BLFS.
- Greg Schafer <greg@linuxfromscratch.org> −− Mainteneur de l'ensemble des outils, éditeur du livre LFS, co−créateur de PLFS.
- Tushar Teredesai <tushar@linuxfromscratch.org> −− Editeur du livre BLFS, Mainteneur des projets Hints (astuces LFS) et Patches (correctifs).
- Jeremy Utley <jeremy@linuxfromscratch.org> −− Editeur du livre LFS, mainteneur Bugzilla.
- Un nombre incroyable d'autres personnes sur les différentes listes de diffusion LFS et BLFS qui ont rendu ce livre possible en apportant leurs suggestions, en testant le livre et en soumettant des rapports de bugs, leurs instructions et leurs expériences pour l'installation des différents packages.

Traducteurs

- Manuel Canales Esparcia <macana@lfs−es.org> −− Projet de traduction espagnol pour LFS.
- Johan Lenglet <johan@linuxfromscratch.org> −− Projet de traduction français pour LFS.
- Anderson Lizardo <lizardo@linuxfromscratch.org> −− Projet de traduction portugais pour LFS.

Mainteneurs des miroirs

- Jason Andrade <jason@dstc.edu.au> −− miroir au.linuxfromscratch.org.
- William Astle <lost@l−w.net> −− miroir ca.linuxfromscratch.org.
- Baque <baque@cict.fr> −− miroir lfs.cict.fr.

- Stephan Brendel <stevie@stevie20.de> −− miroir lfs.netservice−neuss.de.
- Ian Chilton <ian@ichilton.co.uk> −− miroirs us.linuxfromscratch.org, linuxfromscratch.co.uk.
- Fredrik Danerklint <fredan−lfs@fredan.org> −− se.linuxfromscratch.org
- David D.W. Downey <pgpkeys@aeternamtech.com> −− miroir lfs.learnbyexample.com
- Eduardo B. Fonseca <ebf@aedsolucoes.com.br> −− miroir br.linuxfromscratch.org
- Hagen Herrschaft <hrx@hrxnet.de> −− miroir de.linuxfromscratch.org.
- Tim Jackson <tim@idge.net> −− miroir linuxfromscratch.idge.net.
- Barna Koczka <barna@siker.hu> −− miroir hu.linuxfromscratch.org.
- Roel Neefs −− miroir linuxfromscratch.rave.org.
- Simon Nicoll <sime@dot−sime.com> −− miroir uk.linuxfromscratch.org.
- Ervin S. Odisho <ervin@activalink.net> −− miroir lfs.activalink.net.
- Guido Passet <guido@primerelay.net> −− miroir nl.linuxfromscratch.org.
- Mikhail Pastukhov <miha@xuy.biz> −− lfs.130th.net mirror.
- Jeremy Polen <jpolen@rackspace.com> −− miroir us2.linuxfromscratch.org.
- UK Mirror Service −− miroir linuxfromscratch.mirror.co.uk.
- Thomas Skyt <thomas@sofagang.dk> −− miroir dk.linuxfromscratch.org.
- Antonin Sprinzl <Antonin.Sprinzl@tuwien.ac.at> −− miroir at.linuxfromscratch.org.
- Dag Stenstad <dag@stenstad.net> pour avoir apporté no.linuxfromscratch.org et Ian Chilton pour l'avoir fait fonctionner.
- doc.cs.univ− paris8.fr <archive@doc.cs.univ−paris8.fr> −− miroir www2.fr.linuxfromscratch.org.
- Jesse Tie−Ten− Quee <highos@linuxfromscratch.org> pour l'apport et le maintien du serveur linuxfromscratch.org.
- Alexander Velin <velin@zadnik.org> −− miroir bg.linuxfromscratch.org.
- Martin Voss <Martin.Voss@ada.de> −− miroir lfs.linux−matrix.net.
- Pui Yong <pyng@spam.averse.net> −− miroir sg.linuxfromscratch.org.

Donateurs

- Dean Benson <dean@vipersoft.co.uk> pour différentes contributions monétaires.
- DREAMWVR.COM pour leur support en tant que sponsor en donnant différentes ressources pour LFS et ses sous−projets.
- Hagen Herrschaft <hrx@hrxnet.de> pour le don d'un système P4 2,2 Ghz, fonctionnant actuellement sous le nom de *lorien*.
- O'Reilly pour le don de livres sur SQL et PHP.
- VA Software qui, en plus de Linux.com, a donné une station de travail VA Linux 420 (ancien StartX SP2).
- Mark Stone pour avoir donné *shadowfax*, le premier serveur linuxfromscratch.org, un P3 750 MHz avec 512 Mo de RAM et deux disques SCSI de 9 Go. Lorsque le serveur a été déplacé, il a été renommé *belgarath*.
- Jesse Tie−Ten− Quee <highos@linuxfromscratch.org> pour son don d'un graveur de CDRW Yamaha 8824E.
- Un nombre incroyable de personnes sur les différentes listes de diffusion LFS qui ont rendu ce livre possible en apportant leurs suggestions, en testant le livre et en soumettant des rapports de bugs.

Anciens membres de l'équipe et contributeurs

- Timothy Bauscher <timothy@linuxfromscratch.org> −− Editeur du livre LFS, Mainteneur du projet Hints (astuces).
- Robert Briggs pour avoir donné au tout début les noms de domaine *linuxfromscratch.org* et *linuxfromscratch.com*.

- Ian Chilton <ian@ichilton.co.uk> pour maintenir le projet Hints
- Marc Heerdink <gimli@linuxfromscratch.org> −− Editeur du livre LFS.
- Seth W. Klein <sklein@linuxfromscratch.org> −− Créateur de la FAQ LFS.
- Garrett LeSage <garrett@linuxart.com> −− Créateur de la bannière originale de LFS.
- Simon Perreault <nomis80@videotron.ca> −− Mainteneur du projet Hints.
- Geert Poels <Geert.Poels@skynet.be> −− Créateur de la bannière originale de BLFS; basée sur la bannière pour LFS par Garrett LeSage.
- Frank Skettino <bkenoah@oswd.org> pour la conception initiale de l'ancien site web −− jetez un oeil sur OSWD.
- Jesse Tie−Ten− Quee <highos@linuxfromscratch.org> pour avoir répondu à un nombre impressionnant de questions sur IRC et pour avoir une grande patience.

Chapitre 2. Informations importantes

A propos de $LFS

S'il vous plait, lisez ce paragraphe attentivement. Tout au long de ce livre, la variable LFS sera utilisée fréquemment. Vous devrez la remplacer partout où vous la trouverez par le répertoire dans lequel vous avez monté la partition contenant votre système LFS. La méthode pour la créer et l'endroit où monter cette partition seront expliqués en détail au chapitre 3. A titre d'exemple, supposons que la partition LFS soit montée dans le répertoire `/mnt/lfs`.

Quand il vous est demandé de lancer une commande telle que **`./configure --prefix=$LFS/tools`**, il vous faut en pratique exécuter **`./configure --prefix=/mnt/lfs/tools`**.

Il est très important que cela soit fait quel que soit l'endroit où vous le lisez, que ce soit pour une commande à donner à un shell ou dans un fichier édité ou créé.

Une solution possible est de définir la variable d'environnement LFS. De cette façon, $LFS peut être tapé directement au lieu de le remplacer par /mnt/lfs. Cela peut être accomplit en exécutant la commande suivante :

```
export LFS=/mnt/lfs
```

A partir de ce moment, lorsqu'il vous sera demandé d'entrer une commande telle que **`./configure --prefix=$LFS/tools`**,vous pourrez la taper littéralement. Votre shell fera le remplacement de "$LFS" par "/mnt/lfs" lors de l'analyse de la ligne de commande (c'est−à−dire après avoir appuyé sur la touche Entrée).

A propos des SBUs

La plupart des personnes souhaitent savoir combien de temps la compilation et l'installation de chaque package va prendre. Mais, "Linux from Scratch" est construit sur tant de systèmes différents qu'il n'est pas possible de donner des temps précis : le plus gros package (Glibc) ne prendra pas plus de 20 minutes sur les systèmes les plus rapides, mais il prendra environ trois jours sur le moins rapide −− sans plaisanterie. Donc, au lieu de donner les temps exacts, nous avons eu l'idée d'utiliser l'unité Binutils statique (*Static Binutils Unit*) dont l'abréviation est *SBU*.

Cela fonctionne ainsi : le premier package lié statiquement, que vous compilez dans ce livre est Binutils lors du chapitre 5. Le temps que prend la compilation de ce package est ce que nous appelons "SBU". Tous les autres temps de compilation sont exprimés relativement à ce temps.

Par exemple, le temps pris pour construire la version statique de GCC est 4,4 SBU. Ceci signifie que s'il vous a fallu 10 minutes pour compiler et installer le Binutils statique, alors vous savez que cela prendra environ 45 minutes pour construire GCC statique. Heureusement, la plupart des temps de construction sont bien plus court que celui de Binutils.

Notez que si le compilateur système de votre hôte est basé sur GCC−2, les SBU affichés seront quelque peu sous−estimées. Ceci est dû au fait que le SBU est basé sur le tout premier package, compilé avec l'ancien GCC, alors que le reste du système est compilé avec le nouveau GCC−3.3.1 connu pour être environ 30% plus lent.

Notez aussi que les SBU ne fonctionnent pas bien sur les machines SMP. Mais si vous avez la chance d'avoir plusieurs processeurs, il est probable que votre système soit si rapide que vous ne vous en soucierez pas.

A propos des suites de tests

La plupart des packages disposent d'une suite de tests. Lancer cette suite de tests pour un package nouvellement construit est généralement une bonne idée car cela peut apporter une vérification comme quoi tout a été compilé correctement. Une suite de tests réussissant l'ensemble des vérifications prouve généralement que le package fonctionne à peu près comme le développeur en avait l'intention. Néanmoins, cela ne garantit pas que le package ne contient pas de bugs.

Certaines des suites de tests sont plus importantes que d'autres. Par exemple, les suites de tests des packages formant le coeur de l'ensemble des outils −− GCC, Binutils et la bibliothèque C Glibc −− sont de la plus grande importance étant donné leur rôle central dans un système fonctionnel. Mais, faites attention, les suites de tests pour GCC et Glibc peuvent prendre beaucoup de temps pour se terminer, spécialement sur du matériel lent.

Lors de votre progression dans le livre et de votre rencontre avec les commandes de construction lançant les différentes suites de test, nous allons vous guider sur l'importance relative de la suite de tests en question pour que vous puissiez décider vous−même de le lancer ou non.

> **Note :** Un problème commun lors du lancement des suites de test pour Binutils et GCC est de manquer de pseudo−terminaux (en clair des PTY). Le symptôme est un nombre inhabituellement haut de tests ayant échoués. Ceci peut arriver pour un certain nombre de raisons. La plus raisonnable est que le système hôte ne dispose pas d'un système de fichiers *devpts* file configuré correctement. Nous en discuterons plus tard lors du chapitre 5.

Comment demander de l'aide

Si vous rencontrez des problèmes en utilisant ce livre et que votre problème ne figure pas dans la FAQ (http://www.linuxfromscratch.org/faq), vous vous rendrez compte que les personnes sur IRC et sur les listes de discussion sont prêtes à vous aider. Un survol des listes de diffusion de LFS est disponible sur cette page la section intitulée *Listes de diffusion* dans Chapitre 1. Pour nous aider à diagnostiquer et résoudre votre problème, assurez−vous d'inclure autant d'informations utiles que possible dans votre demande d'aide.

Informations de base

A part une brève explication de votre problème, les informations essentielles à inclure dans votre requête sont :

- la version du livre que vous utilisez, c'est−à−dire 5.0,
- la distribution de l'hôte et la version à partir de laquelle vous allez créer LFS,
- avec quel package ou section vous avez un problème,
- quel est le message d'erreur exact, ou quel est le symptôme,
- si vous vous êtes éloigné du livre ou non.

(Notez que dire que vous vous êtes éloigné du livre ne signifie pas que nous ne vous aiderons pas. Après tout, LFS est justement un ensemble de choix. Cela nous aidera simplement à voir les autres causes possibles à votre problème.)

Problèmes de configuration

Quand quelque chose se passe mal pendant l'étape où le script configure est lancé, regardez dans tout le fichier `config.log`. Il contient les erreurs possibles qui ont été rencontrées pendant la phase de configuration, et qui ne sont pas toujours affichées à l'écran. Incluez les lignes appropriées si vous décidez de demander de l'aide.

Problèmes de compilation

Pour nous aider à trouver la cause du problème, à la fois les textes affichés à l'écran et le contenu de plusieurs fichiers sont utiles. Les informations affichées à la fois par le script ./configure et la commande make peuvent être utiles. N'incluez pas aveuglément l'ensemble, mais d'un autre côté n'en donnez pas trop peu. A titre d'exemple, voici l'affichage d'une commande make :

```
gcc -DALIASPATH=\"/mnt/lfs/usr/share/locale:.\"
-DLOCALEDIR=\"/mnt/lfs/usr/share/locale\" -DLIBDIR=\"/mnt/lfs/usr/lib\"
-DINCLUDEDIR=\"/mnt/lfs/usr/include\" -DHAVE_CONFIG_H -I. -I.
-g -O2 -c getopt1.c
gcc  -g -O2 -static -o make  ar.o arscan.o commands.o dir.o expand.o file.o
function.o getopt.o implicit.o job.o main.o misc.o read.o remake.o rule.o
signame.o variable.o vpath.o default.o remote-stub.o version.o opt1.o
-lutil job.o: In function `load_too_high':
/lfs/tmp/make-3.79.1/job.c:1565: undefined reference to `getloadavg'
collect2: ld returned 1 exit status
make[2]: *** [make] Error 1
make[2]: Leaving directory `/lfs/tmp/make-3.79.1'
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory `/lfs/tmp/make-3.79.1'
make: *** [all-recursive-am] Error 2
```

Dans ce genre de cas, nombreuses sont les personnes qui ne fournissent que la section du bas contenant le texte

```
make [2]: *** [make] Error 1
```

jusqu'en bas. Cela n'est pas suffisant pour émettre un diagnostic sur le problème, car cela nous signale que *quelque chose* s'est mal passé, mais pas *ce qui* s'est mal passé. La section complète, comme fournie ci-dessus, constitue ce qui doit être inclus pour nous être utile, car il inclut la commande qui a été exécutée avec son message d'erreur.

Un excellent article sur la façon de demander de l'aide sur Internet en général a été écrit par Eric S. Raymond. Il est disponible en ligne à l'adresse http://catb.org/~esr/faqs/smart-questions.html. Lisez et suivez les astuces indiquées dans ce document et vous aurez plus de chances de recevoir une réponse pour commencer mais aussi d'avoir l'aide dont vous avez actuellement besoin.

Problèmes de la suite de tests

Beaucoup de packages disposent d'une suite de tests que, suivant l'importance du package, nous vous encourageons à lancer. Quelquefois, des packages vont générer des échecs faux ou prévus. Si vous les rencontrez, vous pouvez vérifier la page Wiki de LFS à http://wiki.linuxfromscratch.org./ si nous sommes déjà en train de travailler dessus. Si nous les connaissons déjà, alors il n'est généralement pas nécessaire de se sentir concerné.

II. Deuxième partie − Installation du système LFS

Table des matières

Chapitre 3. Préparer une nouvelle partition

Introduction

Dans ce chapitre, nous allons préparer la partition qui accueillera le système LFS. On créera tout d'abord cette partition, ensuite on y intègrera un système de fichiers puis enfin, on la montera.

Créer une nouvelle partition

Pour construire notre nouveau système Linux, nous allons avoir besoin de place : une partition de disque vide. Si vous n'avez pas de partition libre, et aucune place sur tous vos disques durs pour en créer une, alors vous pouvez créer LFS sur la même partition que celle où votre distribution courante est installée. Cette procédure n'est pas recommandée pour une première installation de LFS, mais si vous êtes en manque de place disque, et que vous êtes courageux, jetez un oeil sur l'astuce http://www.linuxfromscratch.org/hints/downloads/files/lfs_next_to_existing_systems.txt.

Pour un système minimal, vous aurez besoin d'une partition d'environ 1Go. C'est suffisant pour stocker toutes les archives tar et pour compiler tous les packages. Mais si vous avez l'intention d'utiliser le système LFS comme système principal, vous voudrez probablement installer des logiciels supplémentaires et vous aurez alors besoin de plus d'espace, autour de 2 à 3 Go.

Comme nous n'avons jamais assez de RAM dans notre système, c'est une bonne idée d'utiliser une petite partition disque en tant qu'espace d'échange (swap) −− cet espace est utilisé par le noyau pour stocker des données rarement utilisées afin de faire de la place en mémoire pour des choses plus urgentes. La partition swap de votre système LFS peut être la même que celle de votre système hôte, de façon à ce que vous n'ayez pas à en créer une autre si votre système hôte en utilise déjà une.

Démarrez le programme **`cfdisk`** avec comme argument le nom du disque dur sur lequel la nouvelle partition doit être créée −− par exemple `/dev/hda` pour le disque primaire IDE. Créez une partition native Linux, et si nécessaire une partition swap. Référez−vous à la page man de **`cfdisk`** si vous ne savez pas encore utiliser ce programme.

Rappellez−vous la désignation de votre nouvelle partition −− quelque chose comme `hda5`. Ce livre s'y référera comme la partition LFS. Si, maintenant, vous avez aussi une partition swap, rappelez−vous aussi sa désignation. Ces noms seront nécessaires plus tard pour le fichier `/etc/fstab`.

Créer un système de fichiers sur la nouvelle partition

Maintenant que la partition est faite, nous pouvons créer dessus un système de fichiers. Le système de fichiers le plus utilisé dans le monde Linux est le système de fichiers ext2, mais avec les disques durs actuels de grandes capacités, les systèmes de fichiers journalisés deviennent aussi de plus en plus populaires. Ici, nous allons créer un système de fichiers ext2, mais des instructions de construction d'autres systèmes de fichiers peuvent être trouvées sur http://www.linuxfromscratch.org/blfs/view/cvs/postlfs/filesystems.html

Pour créer un système de fichiers ext2 sur la partition LFS, lancez ce qui suit :

```
mke2fs /dev/xxx
```

Remplacer "xxx" par le nom de la partition LFS (quelque chose comme `hda11`).

Si vous avez créé une (nouvelle) partition de swap, vous avez besoin de la réinitialiser aussi comme une partition de swap (aussi connu comme formatage, comme vous avez fait auparavant avec **`mke2fs`**) en lançant :

```
mkswap /dev/yyy
```

Remplacez `yyy` avec le nom de la partition de swap.

Monter la nouvelle partition

Maintenant que nous avons créé un système de fichiers, nous voulons être capable d'accéder à la partition. Pour cela, nous avons besoin de la monter et nous devons choisir un point de montage. Dans ce livre, nous partons du principe que le système de fichiers est monté sous `/mnt/lfs`, mais peu importe le répertoire que vous avez choisi.

Choisissez un point de montage et assignez−le à la variable d'environnement LFS en lançant :

```
export LFS=/mnt/lfs
```

Maintenant, créez le point de montage et montez le système de fichiers LFS en lançant :

```
mkdir −p $LFS
mount /dev/xxx $LFS
```

Remplacez `xxx` avec la désignation de la partition LFS.

Si vous avez décidé d'utiliser de multiples partitions pour LFS (disons une pour `/` et une autre pour `/usr`), montez−les ainsi :

```
mkdir −p $LFS
mount /dev/xxx $LFS
mkdir $LFS/usr
mount /dev/yyy $LFS/usr
```

Bien sûr, remplacez `xxx` et `yyy` avec les noms de partitions appropriés.

Vous devez aussi vous assurer que cette nouvelle partition ne sera pas montée avec des droits trop restrictifs (tels que les options nosuid, nodev ou noatime). Vous pouvez lancer la commande **`mount`** sans paramètre pour voir avec quelles options est montée la partition LFS. Si vous voyez nosuid, nodev ou noatime, vous devrez la remonter.

Maintenant que nous avons créé un endroit pour travailler, nous sommes prêts à télécharger les packages.

Chapitre 4. Les composants : packages et correctifs

Introduction

Ci−dessous, se trouve une liste des packages à télécharger pour construire le système Linux de base. Les numéros de version affichés correspondent à des versions de logiciel *connues* pour fonctionner, et qui sont la base du livre. Sauf si vous êtes un utilisateur LFS expérimenté, nous vous recommendons chaudement de ne pas essayer des versions plus récentes, car les commandes de construction d'une version pourraient ne pas fonctionner avec la dernière version. De même, il y a souvent une bonne raison pour ne pas utiliser la dernière version, généralement à cause de problèmes connus, sans solution pour l'instant.

Toutes les URLs, lorsque c'est possible, se réfèrent aux pages du projet sur http://www.freshmeat.net. Les pages Freshmeat vous donneront un accès simple aux sites officiels de téléchargement ainsi qu'aux sites web des projets, listes de diffusion, journaux de modifications, et plus encore.

Nous ne pouvons pas garantir que ces emplacements de téléchargement seront toujours disponibles. Au cas où un emplacement de téléchargement a changé depuis que ce livre a été publié, merci d'essayer sur google pour trouver le package. Si vous ne le trouvez pas, vous pouvez consulter la page d'erreurs du livre sur http://linuxfromscratch.org/lfs/print ou, mieux encore, d'essayer les autres moyens de téléchargement listés sur http://linuxfromscratch.org/lfs/packages.html

Vous aurez besoin de stocker tous les packages et correctifs téléchargés dans un endroit pratique pendant toute la construction. Vous aurez aussi besoin d'un répertoire dans lequel décompacter les sources et les construire. Un schéma fonctionnant correctement est d'utiliser `$LFS/sources` comme emplacement de stockage des archives tar et des correctifs *mais aussi* comme répertoire de travail. De cette façon, tout ce dont vous avez besoin se situe dans la partition LFS et disponible lors de toutes les étapes du processus de construction.

Donc, vous pouvez vouloir exécuter, en tant que *root*, la commande suivante avant de commencer votre session de téléchargement :

```
mkdir $LFS/sources
```

et rendre ce répertoire modifiable pour votre utilisateur habituel −− car nous supposons que vous n'allez pas télécharger en tant que *root* :

```
chmod a+wt $LFS/sources
```

Tous les packages

Téléchargez ou obtenez d'une autre façon les packages suivants :

Autoconf (2.57) − 792 Ko : http://freshmeat.net/projects/autoconf/ Automake (1.7.6) − 545 Ko : http://freshmeat.net/projects/automake/ Bash (2.05b) − 1 910 Ko : http://freshmeat.net/projects/gnubash/ Binutils (2.14) − 10 666 Ko : http://freshmeat.net/projects/binutils/ Bison (1.875) − 796 Ko : http://freshmeat.net/projects/bison/ Bzip2 (1.0.2) − 650 Ko : http://freshmeat.net/projects/bzip2/ Coreutils (5.0) − 3.860 Ko : http://freshmeat.net/projects/coreutils/ DejaGnu (1.4.3) − 1 775 Ko : http://freshmeat.net/projects/dejagnu/ Diffutils (2.8.1) − 762 Ko : http://freshmeat.net/projects/diffutils/

E2fsprogs (1.34) − 3 003 Ko : http://freshmeat.net/projects/e2fsprogs/ Ed (0.2) − 182 Ko : http://freshmeat.net/projects/ed/ Expect (5.39.0) − 508 Ko : http://freshmeat.net/projects/expect/ File (4.04) − 338 Ko : (*) voir la note ci−dessous http://freshmeat.net/projects/file/ Findutils (4.1.20) − 760 Ko : http://freshmeat.net/projects/findutils/ Flex (2.5.4a) − 372 Ko : ftp://ftp.gnu.org/gnu/non−gnu/flex/ Gawk (3.1.3) − 1 596 Ko : http://freshmeat.net/projects/gnuawk/ GCC (2.95.3) − 9 618 Ko : http://freshmeat.net/projects/gcc/ GCC−core (3.3.1) − 10 969 Ko : http://freshmeat.net/projects/gcc/ GCC−g++ (3.3.1) − 2 017 Ko : http://freshmeat.net/projects/gcc/ GCC−testsuite (3.3.1) − 1 033 Ko : http://freshmeat.net/projects/gcc/ Gettext (0.12.1) − 5 593 Ko : http://freshmeat.net/projects/gettext/ Glibc (2.3.2) − 13 064 Ko : http://freshmeat.net/projects/glibc/ Glibc−linuxthreads (2.3.2) − 211 Ko : http://freshmeat.net/projects/glibc/ Grep (2.5.1) − 545 Ko : http://freshmeat.net/projects/grep/ Groff (1.19) − 2 360 Ko : http://freshmeat.net/projects/groff/ Grub (0.93) − 870 Ko : ftp://alpha.gnu.org/pub/gnu/grub/ Gzip (1.3.5) − 324 Ko : ftp://alpha.gnu.org/gnu/gzip/ Inetutils (1.4.2) − 1 019 Ko : http://freshmeat.net/projects/inetutils/ Kbd (1.08) − 801 Ko : http://freshmeat.net/projects/kbd/ Less (381) − 259 Ko : http://freshmeat.net/projects/less/ LFS−Bootscripts (1.12) − 25 Ko : http://downloads.linuxfromscratch.org/lfs−bootscripts−1.12.tar.bz2 Lfs−Utils (0.3) − 221 Ko : http://www.linuxfromscratch.org/~winkie/downloads/lfs−utils/ Libtool (1.5) − 2.751 Ko : http://freshmeat.net/projects/libtool/ Linux (2.4.22) − 28 837 Ko : http://freshmeat.net/projects/linux/ M4 (1.4) − 310 Ko : http://freshmeat.net/projects/gnum4/ Make (3.80) − 899 Ko : http://freshmeat.net/projects/gnumake MAKEDEV (1.7) − 8 Ko : http://downloads.linuxfromscratch.org/MAKEDEV−1.7.bz2 Man (1.5m2) − 196 Ko : http://freshmeat.net/projects/man/ Man−pages (1.60) − 627 Ko : http://freshmeat.net/projects/man−pages/ Modutils (2.4.25) − 215 Ko : http://freshmeat.net/projects/modutils/ Ncurses (5.3) − 2 019 Ko : http://freshmeat.net/projects/ncurses/ Net−tools (1.60) − 194 Ko : http://freshmeat.net/projects/net−tools/ Patch (2.5.4) − 182 Ko : http://freshmeat.net/projects/patch/ Perl (5.8.0) − 10 765 Ko : http://freshmeat.net/projects/perl/ Procinfo (18) − 24 Ko : http://freshmeat.net/projects/procinfo/ Procps (3.1.11) − 242 Ko : http://freshmeat.net/projects/procps/ Psmisc (21.3) − 259 Ko : http://freshmeat.net/projects/psmisc/ Sed (4.0.7) − 678 Ko : http://freshmeat.net/projects/sed/ Shadow (4.0.3) − 760 Ko : http://freshmeat.net/projects/shadow/ Sysklogd (1.4.1) − 80 Ko : http://freshmeat.net/projects/sysklogd/ Sysvinit (2.85) − 91 Ko : http://freshmeat.net/projects/sysvinit/ Tar (1.13.25) − 1 281 Ko : ftp://alpha.gnu.org/gnu/tar/ Tl (8.4.4) − 3 292 Ko : http://freshmeat.net/projects/tcltk/ Texinfo (4.6) − 1 317 Ko : http://freshmeat.net/projects/texinfo/ Util−linux (2.12) − 1 814 Ko : http://freshmeat.net/projects/util−linux/ Vim (6.2) − 3 193 Ko : http://freshmeat.net/projects/vim/ Zlib (1.1.4) − 144 Ko : http://freshmeat.net/projects/zlib/
Total size of these packages: 134 Mo ;

> **Note :** File (4.04) pourrait ne pas être disponible au moment où vous lirez ceci. L'emplacement de téléchargement principal est connu pour supprimer les anciennes versions lorsque de nouvelles arrivent. Merci de vous référer à la section correspondante dans Annexe A pour un autre emplacement de téléchargement.

Correctifs nécessaires

En plus de tous ces packages, vous aurez aussi besoin de plusieurs correctifs. Ils corrigent de petites erreurs dans les packages et devraient être intégrés par le mainteneur, ou sont de simples modifications pour aller dans notre sens. Vous aurez besoin des suivants :

Correctif Bash − 7 Ko : http://www.linuxfromscratch.org/patches/lfs/5.0/bash−2.05b−2.patch
Correctif Bison Attribute − 2 Ko : http://www.linuxfromscratch.org/patches/lfs/5.0/bison−1.875−attribute.patch

Correctif Coreutils Hostname − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−hostname−2.patch
Correctif Coreutils Uname − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−uname.patch Correctif Ed Mkstemp − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/ed−0.2−mkstemp.patch Correctif Expect Spawn − 6 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/expect−5.39.0−spawn.patch
Correctif Gawk Libexecdir − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/gawk−3.1.3−libexecdir.patch
Correctif GCC No−Fixincludes − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−no_fixincludes−2.patch
Correctif GCC Specs − 10 Ko : http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−specs−2.patch
Correctif GCC Suppress−Libiberty − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−suppress−libiberty.patch
Correctif GCC−2 − 16 Ko : http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−2.patch
Correctif GCC−2 No−Fixincludes − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−no−fixinc.patch
Correctif GCC−2 Return−Type − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−returntype−fix.patch
Correctif Glibc Sscanf − 2 Ko : http://www.linuxfromscratch.org/patches/lfs/5.0/glibc−2.3.2−sscanf−1.patch
Correctif Grub Gcc33 − 1 Ko : http://www.linuxfromscratch.org/patches/lfs/5.0/grub−0.93−gcc33−1.patch
Correctif Kbd More−Programs − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/kbd−1.08−more−programs.patch
Correctif Man 80−Columns − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−80cols.patch Correctif Man Manpath − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−manpath.patch Correctif Man Pager − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−pager.patch Correctif Ncurses Etip − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−etip−2.patch
Correctif Ncurses Vsscanf − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−vsscanf.patch
Correctif Net−tools Mii−Tool−Gcc33 − 2 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/net−tools−1.60−miitool−gcc33−1.patch
Correctif Perl Libc − 1 Ko : http://www.linuxfromscratch.org/patches/lfs/5.0/perl−5.8.0−libc−3.patch
Correctif Procps Locale − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/procps−3.1.11−locale−fix.patch
Correctif Shadow Newgrp − 1 Ko :
http://www.linuxfromscratch.org/patches/lfs/5.0/shadow−4.0.3−newgrp−fix.patch
Correctif Zlib Vsnprintf − 10 Ko : http://www.linuxfromscratch.org/patches/lfs/5.0/zlib−1.1.4−vsnprintf.patch

En plus des correctifs requis ci−dessus, il en existe un certain nombre d'autres, optionnels, créés par la communauté LFS. La plupart d'entre eux résolvent des problèmes légers ou activent certaines fonctionnalités non disponibles par défaut. Vous être libre d'examiner la base de données des correctifs, située sur http://www.linuxfromscratch.org/patches/ et de récupérer tout correctif supplémentaire qui vous intéresse.

Chapitre 5. Préparer le système LFS

Introduction

Dans ce chapitre, nous allons compiler et installer un système Linux minimal. Ce système ne contiendra que les outils nécessaires pour être capable de commencer la construction du système LFS final dans le prochain chapitre.

La construction de ce système minimal est fait en deux étapes : tout d'abord nous construisons un ensemble d'outils tous nouveaux et indépendant de l'hôte (compilateur, assembleur, éditeur de liens et bibliothèques) et ensuite nous l'utilisons pour construire tous les autres outils essentiels.

Les fichiers compilés dans ce chapitre vont être installés sous le répertoire `$LFS/tools`, de façon à les garder séparés des fichiers installés dans le chapitre suivant. Comme tous les packages compilés ici sont simplement temporaires, nous ne voulons pas polluer le futur système LFS.

La clef pour apprendre ce qui fait fonctionner un système Linux est de savoir exactement quelle est l'utilité de chaque package et pourquoi l'utilisateur ou le système en a besoin. Pour cette raison, une courte description du contenu de chaque package est fournie juste avant les instructions d'installation. Pour une courte description de chaque programme dans un package, merci de vous référer à la section correspondante dans Annexe A.

Les instructions de construction supposent que vous utilisez le shell bash. Il est aussi attendu que vous avez déjà déballé les sources pour un package et êtes entré (**cd**) dans le répertoire des sources avant de lancer les commandes de construction.

Plusieurs packages sont corrigés avant d'être compilés, mais seulement dans le cas où la correction est nécessaire pour résoudre un problème. Souvent, le correctif est nécessaire à la fois dans ce chapitre et dans le suivant, mais quelque fois dans seulement un des deux. Donc, ne vous inquiétez pas lorsque des instructions pour un correctif téléchargé semblent manquer.

Pendant l'installation de la plupart des packages, vous verrez, très probablement, toutes sortes de messages d'avertissement du compilateur qui défilent sur votre écran. Ceci est normal et peut être ignoré sans danger. Il s'agit seulement de messages d'avertissement −− principalement au sujet d'utilisation obsolète, mais pas invalide, de la syntaxe de C ou de C++. Ceci est dû au fait que les standards C ont changé assez souvent et que quelques packages continuent à utiliser les anciens standards ; ce qui n'est pas un véritable problème.

Sauf cas contraire, vous devriez habituellement supprimer les répertoires des sources et de construction après l'installation de chaque package −− tout simplement pour faire du ménage et conserver ainsi de l'espace disque.

Avant de continuer, assurez−vous que la variable LFS est correctement définie en lançant ce qui suit :

```
echo $LFS
```

Assurez−vous que le résultat contient le bon répertoire vers le point de montage de la partition LFS, qui est `/mnt/lfs` si vous avez suivi notre exemple.

```
echo $LFS
```

Notes techniques sur l'atelier d'outils

Cette section essaie d'expliquer quelques détails techniques et logiques concernant la méthode de construction. Il n'est pas essentiel de tout comprendre immédiatement. La plupart des explications prendra du sens une fois que vous aurez réalisé une construction complète. Vous pouvez aussi vous y référer plus tard.

Le but global du chapitre 5 est de construire un environnement sain et temporaire dans lequel nous pouvons entrer avec chroot et à partir duquel nous pouvons constuire un système LFS propre, sans problème lors du chapitre 6. Pou le moment, nous essayons de nous affranchir du système hôte autant que possible en créant un atelier (ensemble d'outils) qui se suffit à lui−même. Il doit être noté que le processus de construction a été construit de telle façon que les risques sont minimisés pour les nouveaux lecteurs et qu'il apporte une valeur éducative maximum en même temps. En d'autres termes, des techniques plus avancées pourraient être utilisées pour construire le système.

> **Important :** Avant de continuer, vous devez réellement savoir le nom de votre plate−forme actuelle, souvent aussi appelé le *triplé cible*. Pour beaucoup de personnes, ce triplé sera, par exemple : *i686−pc−linux−gnu*. Une façon simple de déterminer votre triplé cible est de lancer le script `config.guess` fourni avec le source de nombreux packages. Déballez les sources de Binutils, lancez le script **`./config.guess`** et notez la sortie.
>
> Vous aurez aussi besoin de connaître le nom de l'*éditeur de liens dynamiques* de votre plateforme, aussi appelé *chargeur dynamique*, à ne pas confondre avec l'éditeur de liens standard *ld* faisant partie de Binutils. L'éditeur de liens dynamique est fourni par Glibc et a pour but de trouver et charger les bibliothèques dynamiques nécessaires à un programme, préparant l'exécution de ce programme et le lançant. Pour la plupart des personnes, le nom de l'éditeur de liens dynamique ser *ld−linux.so.2*. Sur des plateformes moins communes, le nom pourrait être *ld.so.1* et pour les nouvelles plateformes 64 bit, cela pourrait être complètement différent. Vous devez être capable de déterminer le nom de l'éditeur de liens dynamiques de votre plateforme en regardant dans le répertoire `/lib` de votre système hôte. Vous pouvez inspecter un binaire quelconque sur votre système hôte en lançant **`'readelf −l <name of binary> | grep interpreter'`** et en notant la sortie. La référence concernant toutes les plateformes est dans le fichier `shlib−versions` à la racine du répertoires des sources de Glibc.

Quelques points techniques clés sur la façon dont fonctionne la méthode de construction au chapitre 5 :

- Similaire en principe à la cross−compilation où des outils installés avec le même préfixe fonctionnent en coopération et donc utilisent un peu de "magie" GNU.
- Des modifications attentionnées du chemin de recherche des bibliothèques de l'éditeur de liens standard pour vous assurer que les programmes sont liés uniquement avec les bibliothèques que nous avons choisi.
- Des modifications attentionnées du fichier *specs* de **`gcc`** pour indiquer au compilateur où l'éditeur de liens dynamique cible sera utilisé.

Binutils est installé en premier parce que GCC et Glibc réalisent des tests de fonctionnalités sur l'assembleur et l'éditeur de liens durant leur lancement respectif de **`./configure`** pour déterminer les fonctionnalités (dés)activées pour le logiciel. Ceci est plus important que ce que vous pouvez imaginer. Un GCC ou Glibc mal configurés peuvent avoir pour résultat un ensemble d'outils subtilement bogués et l'impact d'une telle erreur n'apparaîtra qu'à la fin de la construction de la distribution complète. Heureusement, un échec de la suite de tests nous alertera habituellement avant que trop de temps ne se soit passé.

Binutils installe son assembleur et son éditeur de liens en deux emplacements, `/tools/bin` et `/tools/$TARGET_TRIPLET/bin`. En fait, les outils dans un emplacement sont des liens vers les autres. Une importante facette de l'éditeur de liens réside dans l'ordre de son chemin de recherche des bibliothèques. Des informations détaillées sont disponibles sur **ld** en lui passant l'option *--verbose*. Par exemple, **'ld --verbose | grep SEARCH'** vous montrera le chemin actuel et leur ordre. Vous pouvez voir quels fichiers sont actuellement liés par **ld** en compilant un programme et en ajoutant l'option *--verbose* switch. Par exemple, **'gcc dummy.c -Wl,--verbose 2>&1 | grep succeeded'** vous affichera tous les fichiers ouverts avec succès lors du lien.

Le prochain package installé est GCC et, lors de exécution de **./configure**, vous verrez par exemple :

```
checking what assembler to use... /tools/i686-pc-linux-gnu/bin/as
checking what linker to use... /tools/i686-pc-linux-gnu/bin/ld
```

Ceci est important pour les raisons mentionnées ci-dessus. Cela démontre aussi que le script configure de GCC ne cherche pas dans les répertoires $PATH pour trouver les outils à utiliser. Néanmoins, lors des opérations sur **gcc** lui-même, les mêmes chemins de recherche ne sont pas nécessairement utilisés. Vous pouvez trouver quel éditeur de liens standard **gcc** va utiliser en lançant : **'gcc -print-prog-name=ld'**. Des informations détaillées peuvent être obtenues à partir de **gcc** en lui passant l'option *-v* lors de la compilation d'un programme. Par exemple : **'gcc -v dummy.c'** vous affichera des informations sur le préprocesseur, les étapes de compilation et d'assemblage en incluant les chemins de recherche des en-têtes de **gcc** ainsi que leur ordre.

Le prochain paquet installé est Glibc. L'attention la plus importante, lors de la compilation de Glibc, doit être portée sur le compilateur et les outils. Le compilateur n'est généralement pas un problème car Glibc utilise toujours le **gcc** trouvé dans un répertoire de $PATH. Les outils binaires et les en-têtes du noyau peuvent poser plus de problèmes. Donc, nous ne prenons aucun risque et utilisons les options disponibles par configure. Après le lancement de **./configure**, vous pouvez vérifier le contenu du fichier `config.make` du répertoire `glibc-build` pour tous les détails importants. Vous noterez des éléments intéressants comme l'utilisation de **CC="gcc -B/tools/bin/"** pour contrôler quels outils binaires sont utilisés, mais aussi l'utilisation des options *-nostdinc* et *-isystem* pour contrôler le chemin de recherche des fichiers include du compilateur. Ces éléments aident à souligner un aspect important du package Glibc : il est tout à fait suffisant en terme de machinerie de construction et ne repose pas en général sur les valeurs par défaut de l'ensemble des outils.

Après l'installation de Glibc, nous faisons quelques ajustements pour nous assurer que la recherche et l'édition de liens prennent place dans notre préfixe `/tools`. Nous installons un **ld** personnalisé, disposant d'un chemin de recherche en dur vers `/tools/lib`. Ensuite, nous modifions le fichier specs de **gcc** pour pointer vers notre éditeur de liens dynamique dans `/tools/lib`. Cette dernière étape est *vitale* pour tout le processus. Comme mentionné ci-dessus, un chemin codé en dur vers un éditeur de liens dynamiques est embarqué dans chaque exécutable ELF. Vous pouvez l'inspecter en lançant : **'readelf -l <name of binary> | grep interpreter'**. En modifiant le fichier specs de **gcc**nous nous assurons que tout programme compilé ici (et jusqu'à la fin du chapitre 5) utilisera notre nouvel éditeur de liens compris dans `/tools/lib`.

La nécessité d'utiliser le nouvel éditeur de liens dynamiques est aussi la raison pour laquelle nous appliquons le correctif Specs lors de la deuxième passe pour GCC. Son échec aura pour résultat des programmes GCC comprenant le nom de l'éditeur de liens du répertoire `/lib` sur le système hôte, ce qui ferait échouer notre but de s'éloigner de l'hôte.

Lors de la deuxième passe pour Binutils, nous sommes capable d'utiliser l'option configure *--with-lib-path* pour contrôler le chemin de recherche de la bibliothèque de **ld**. A partir de ce moment, notre atelier principal d'outils se suffit à lui-même. Le reste des packages du chapitre 5 sera construit avec le répertoire `/tools`.

En entrant dans l'environnement chroot chapitre 6, le premier paquet majeur que nous installons est Glibc, en raison de sa nature auto suffisante. Une fois que cette Glibc est installée dans `/usr`, nous réalisons un rapide changement dans les paramètres par défaut de l'atelier des outils, puis procédons à la construction du reste de la cible chapitre 6.

Notes sur l'édition de liens statiques

La plupart des programmes doivent réaliser, en plus de leur tâches spécifiques, beaucoup d'opérations communes et quelques fois triviales. Cela inclut l'allocation de mémoire, la recherche dans des répertoires, la lecture et l'écriture de fichiers, la gestion de chaînes de caractères, la correspondance de modèles, l'arithmétique et bien d'autres tâches. Au lieu d'obliger tout programme à réinventer la roue, le système GNU apporte toutes les fonctions de base dans des bibliothèques toutes prêtes. La bibliothèque principale sur tout système Linux est *Glibc*.

Il existe principalement deux façons de lier les fonctions d'une bibliothèque à un programme qui les utilise : statiquement ou dynamiquement. Quand un programme est lié statiquement, le code des fonctions utilisées est inclut dans l'exécutable, générant ainsi un programme lourd. Lorsqu'un programme est lié dynamiquement, ce qui est inclu est une référence de l'éditeur de liens, du nom de la bibliothèque et du nom de la fonction, résultant en un exécutable bien plus petit. (Une troisième façon est d'utiliser l'interface de programmation de l'éditeur de liens. Voir la page man *dlopen* pour plus d'informations.)

L'édition de liens dynamiques est réalisée par défaut sur Linux et a trois avantages majeurs sur l'édition de liens statiques. Tout d'abord, vous n'avez besoin que d'une copie du code de la bibliothèque exécutable sur votre disque dur au lieu d'avoir plusieurs copies du même code inclut dans un grand nombre de programmes, donc en sauvant de l'espace disque. Deuxièmement, quand plusieurs programmes utilisent la même fonction d'une bibliothèque en même temps, seule une copie de la fonction de la bibliothèque est requise dans le coeur, ce qui sauve de l'espace mémoire. Enfin, quand une bibliothèque de fonction est déboguée ou améliorée, vous n'avez besoin que de recompiler cette bibliothèque, au lieu d'avoir à recompiler tous les programmes utilisant la fonction améliorée.

Si l'édition de liens a plusieurs avantages, alors pourquoi lier statiquement les deux premiers packages de ce chapitre ? Il existe trois raisons : historique, éducationnel et technique. Historique parce que les anciennes versions de LFS liaient statiquement tous les programmes de ce chapitre. Educationnel parce que connaître la différence est utile. Technique parce que nous gagnons un élément d'indépendance sur l'hôte. Néanmoins, il est bon de rappeler qu'une construction complète et réussie de LFS peut toujours s'achever lorsque les deux premiers packages sont liés dynamiquement.

Créer le répertoire $LFS/tools

Tous les programmes compilés dans ce chapitre seront installés sous `$LFS/tools` pour les séparer des programmes compilés dans le prochain chapitre. Les programmes compilés ici ne sont que des outils temporaires et ne feront pas partie du système LFS final. En les conservant à part, nous pourrons plus facilement les supprimer plus tard.

Si vous souhaitez plus tard chercher dans les binaires de votre système pour voir quels fichiers ils utilisent ou à quels fichiers ils sont liés, alors, pour rendre cette recherche plus simple, vous pouvez choisir un nom

unique. Au lieu du simplissime "tools", vous pouvez utiliser quelque chose comme "tools−for−lfs".

Créez le répertoire requis en lançant ce qui suit :

```
mkdir $LFS/tools
```

La prochaine étape consiste en la création d'un lien symbolique `/tools` sur l'hôte système. Il pointera vers le répertoire que nous venons de créer sur la partition LFS :

```
ln −s $LFS/tools /
```

Ce lien symbolique nous permet de compiler notre ensemble d'outils de façon à ce qu'il se réfère en permanence à `/tools`, ce qui signifie que le compilateur, l'assembleur et l'éditeur de liens fonctionneront dans ce chapitre (alors que nous sommes encore en train d'utiliser quelques outils de l'hôte) *et* dans le prochain (quand nous serons entré dans le chroot de la partition LFS).

> **Note :** Etudiez la commande ci−dessus attentivement. Elle peut être étonnante à première vue. La commande **`ln`** dispose d'une syntaxe variable, donc assurez−vous de vérifier la page man de ln avant d'indiquer qu'elle est erronée.

Ajouter l'utilisateur lfs

Lorsque vous êtes connecté en tant que *root*, faire une seule erreur peut endommager ou même bloquer votre serveur. Donc, nous vous recommandons de construire les package de ce chapitre en tant qu'utilisateur non privilégié. Vous pouvez utiliser votre propre nom d'utilisateur, mais pour rendre plus facile la préparation d'un environnement de travail sain, nous allons créer un nouvel utilisateur *lfs* et utiliser celui−ci lors du processus d'installation. En tant que *root*, lancez les commandes suivantes pour ajouter un nouvel utilisateur :

```
useradd −s /bin/bash −m lfs
passwd lfs
```

Maintenant, donnez à ce nouvel utilisateur *lfs* tous les droits sur le répertoire `$LFS/tools` en l'enregistrant comme propriétaire du fichier :

```
chown lfs $LFS/tools
```

Si vous utilisez un répertoire de travail partagé, faites que l'utilisateur *lfs* en soit le propriétaire:

```
chown lfs $LFS/sources
```

Ensuite, connectez−vous en tant qu'utilisateur *lfs*. Ceci peut être fait via une console virtuelle, un gestionnaire d'affichage ou avec la commande de substitution d'utilisateur :

```
su − lfs
```

Le "**−**" indique à **`su`** de lancer un tout nouveau shell.

Configurer l'environnement

Une fois connecté en tant qu'utilisateur *lfs*, lancez les commandes suivantes pour mettre en place un bon environnement de travail :

```
cat > ~/.bash_profile << "EOF"
set +h
umask 022
LFS=/mnt/lfs
LC_ALL=POSIX
LDFLAGS="-s"
PATH=/tools/bin:$PATH
export LFS LC_ALL LDFLAGS PATH
unset CC CXX CPP LD_LIBRARY_PATH LD_PRELOAD
EOF
source ~/.bash_profile
```

La commande **`set +h`** arrête la fonction de hachage de **`bash`**. Le hachage est normalement une fonction utile. **`bash`** utilise une table hachée pour se rappeler les chemins complets des fichiers exécutables ce qui évite de nombreuses recherches dans 'PATH' à chaque fois qu'un éxécutable doit être exécuté. Néanmoins, nous souhaitons que les nouveaux outils deviennent disponibles tout de suite après leur installation. En désactivant la fonction de hachage, nos commandes "interactives" (**`make`**, **`patch`**, **`sed`**, **`cp`**, etc.) utiliseront toujours les dernières versions disponibles lors du processus de construction.

Ce profil configure le umask à 022, donc les fichiers et les répertoires nouvellement créés auront les bonnes permissions. Pour être plus précis, seul le propriétaire du fichier aura la permission d'écriture sur les nouveaux fichiers et répertoires. Les autres utilisateurs du système auront le droit de lire et le droit de traverser les répertoires. Il est conseillé de garder ce paramétrage tout au long de l'installation LFS.

La variable LFS doit bien sûr être configuré au point de montage que vous avez choisi.

La variable LC_ALL contrôle l'emplacement de certains programmes, en faisant que certains messages suivent les conventions du pays spécifié. Si votre système hôte utilise une version de la Glibc plus ancienne que la 2.2.4, avoir LC_ALL positionné sur quelque chose d'autre que "POSIX" ou "C" durant ce chapitre peut poser des problèmes si vous sortez de l'environnement chroot et souhaitez y retourner plus tard. En positionnant LC_ALL à "POSIX" (ou "C", les deux étant équivalents), nous nous assurons que tout fonctionnera comme attendu dans l'environnement chroot.

Nous ajoutons `/tools/bin` au début du PATH standard de façon à ce que, au fur et à mesure de notre progression dans ce chapitre, les outils que nous avons construits soient utilisés lors du reste du processus de construction.

Les variables d'environnement CC, CXX, CPP, LD_LIBRARY_PATH et LD_PRELOAD peuvent toutes potentiellement poser problème avec notre ensemble d'outils du chapitre 5 Nous devons donc les dés−initialiser pour supprimer tout risque que cela arrive.

Maintenant, après avoir intégré les profiles tout récemment créés, nous avons complètement terminé la configuration et la construction des outils temporaires qui nous aideront dans les chapitres qui suivent.

Installer Binutils−2.14 − Pass 1

```
Estimation du temps de construction :   1,0 SBU
Estimation de l'espace disque requis :  194 Mo
```

Contenu de Binutils

Binutils est une collection d'outils de développement de logiciels contenant un éditeur de liens, un assembleur et d'autres outils pour travailler avec des fichiers objet et des archives.

Programmes installés : addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings et strip

Bibliothèques installés : libiberty.a, libbfd.[a,so] et libopcodes.[a,so]

Dépendances d'installation de Binutils

Binutils dépend de Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Installation de Binutils

Il est important que Binutils soit le premier package à être compilé parce que à la fois Glibc et GCC réalisent différents tests sur l'éditeur de liens et l'assembleur disponibles pour déterminer leur propres fonctionnalités à activer.

> **Note :** Même si Binutils est un package important de l'ensemble des outils, nous n'allons pas lancer la suite de tests aussi tôt. Tout d'abord, les éléments de la suite de tests ne sont pas encore en place. Ensuite, les programmes de la première passe seront bientôt écrasés par ceux de la seconde passe.

Ce package est connu pour mal se comporter si vous changez les options d'optimisation par défaut (en incluant les options −march et −mcpu). Donc, si vous avez défini des variables d'environnement qui surchargent les optimisations pas défaut, telles que CFLAGS et CXXFLAGS, nous vous recommandons de supprimer cette initialisation lors de la construction de Binutils.

La documentation de Binutils recommande de construire Binutils en dehors du répertoire des sources, c'est−à−dire dans un répertoire de construction dédié :

```
mkdir ../binutils-build
cd ../binutils-build
```

> **Note :** Si vous voulez que les valeurs SBU listées dans le reste du livre vous soient utiles, vous devez mesurer le temps pris pour construire ce package. Pour cela, vous pouvez faire quelque chose comme ça : **`time { ./configure ... && ... && ... && make install; }`**.

Maintenant, préparez la compilation de Binutils :

```
../binutils-2.14/configure \
    --prefix=/tools --disable-nls
```

La signification des options de configure est :

- **`--prefix=/tools`** : Ceci indique au script configure de se préparer à installer les programmes Binutils dans le répertoire `/tools`.
- **`--disable-nls`** : Ceci désactive l'internationalisation (un mot généralement abrégé en i18n). Nous n'en avons pas besoin pour nos programmes statiques et *nls* cause souvent des problèmes lors d'une édition des liens en statique.

Continuez avec la compilation du package :

```
make configure-host
make LDFLAGS="-all-static"
```

La signification des paramètres de make est :

- **`configure-host`** : Ceci force tous les sous−répertoires à être immédiatement configurés. Une construction statique échouera sans ça. Nous utilisons donc cette option pour contourner le problème.
- **`LDFLAGS="-all-static"`** : Ceci indique à l'éditeur de liens que tous les programmes Binutils devraient être liés statiquement. Néanmoins, en étant strict, **`"-all-static"`** est tout d'abord passé au programme *libtool* qui passe ensuite **`"-static"`** à l'éditeur de liens.

Enfin, installez le package :

```
make install
```

Maintenant, préparez l'éditeur de liens pour le "verrouillage" de Glibc un peu plus tard :

```
make -C ld clean
make -C ld LDFLAGS="-all-static" LIB_PATH=/tools/lib
```

La signification des paramètres de make est :

- **`-C ld clean`** : Ceci indique au programme make de supprimer tous les fichiers compilés mais seulement dans le répertoire `ld`.
- **`-C ld LDFLAGS="-all-static" LIB_PATH=/tools/lib`** : Cette option reconstruit tout ce qui se trouve dans le sous−répertoire `ld`. Spécifier la variable makefile LIB_PATH en ligne de commande nous autorise à écraser la valeur par défaut et à la faire pointer vers notre emplacement temporaire des outils. La valeur de cette variable spécifie le chemin de recherche des bibliothèques par défaut pour l'éditeur de liens. Vous verrez comment cette préparation est ensuite utilisée dans le chapitre.

Avertissement
Ne supprimez pas encore les répertoires de construction et des sources dont nous aurons encore besoin dans leur état actuel un peu plus tard dans ce chapitre.

Installer GCC−3.3.1 − Pass 1

```
Estimation du temps de construction :   4,4 SBU
Estimation de l'espace disque requis :  300 Mo
```

Contenu de GCC

Le package GCC contient le compilateur GNU, incluant les compilateurs C et C++.

Programmes installés : c++, cc (lien vers gcc), cc1, cc1plus, collect2, cpp, g++, gcc, gccbug et gcov

Bibliothèques installées : libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.[a,so] et libsupc++.a

Dépendances d'installation de GCC

GCC dépend de Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Installation de GCC

Déballez seulement l'archive tar GCC−core, car nous n'allons pas avoir besoin d'un compilateur C++ pour l'instant.

> **Note :** Même si GCC est un package important de l'ensemble des outils, nous n'allons pas lancer la suite de tests aussi tôt. Tout d'abord, les éléments de la suite de tests ne sont pas encore en place. Ensuite, les programmes de la première passe seront bientôt écrasés par ceux de la seconde passe.

Ce package est connu pour mal se comporter si vous changez les options d'optimisation par défaut (en incluant les options −march et −mcpu). Donc, si vous avez défini des variables d'environnement qui surchargent les optimisations pas défaut, telles que CFLAGS et CXXFLAGS, nous vous recommandons de supprimer cette initialisation lors de la construction de GCC.

La documentation de GCC recommande de ne pas construire GCC dans le répertoire des sources mais dans un répertoire de construction dédié :

```
mkdir ../gcc-build
cd ../gcc-build
```

Préparez la compilation de GCC :

```
../gcc-3.3.1/configure --prefix=/tools \
    --with-local-prefix=/tools \
    --disable-nls --enable-shared \
    --enable-languages=c
```

La signification des options de configure est :

- **--with-local-prefix=/tools** : Le but de cette option est de supprimer `/usr/local/include` du chemin de recherche des fichiers include de **gcc**. Ce n'est pas absolument essentiel ; néanmoins, vous voulons minimiser l'influence du système hôte, et ceci nous semble un point intéressant à faire dans ce but.
- **--enable-shared** : Cette option peut ne pas sembler intuitive au début. Mais en l'utilisant, elle permet la construction de `libgcc_s.so.1` et `libgcc_eh.a`, et disposer de `libgcc_eh.a` nous assure que le script configure de Glibc (le prochain package à compiler) produira de bons résultats. Notez que les binaires **gcc** seront toujours liés statiquement, car ceci est contrôlé par la valeur **-static** de BOOT_LDFLAGS.
- **--enable-languages=c** : Cette option nous assure que seul le compilateur C sera construit. Cette option n'est nécessaire que si vous avez téléchargé et déballé l'archive tar GCC complète.

Continue avec la compilation du package :

```
make BOOT_LDFLAGS="-static" bootstrap
```

La signification des paramètres de make est :

- **BOOT_LDFLAGS="-static"** : Ceci indique à GCC de lier ses programmes statiquement.
- **bootstrap** : Cette cible ne compile pas GCC une seule fois mais plusieurs fois. Il utilise les programmes compilés dans le premier tour pour se compiler une deuxième fois, puis une troisième fois. Il compare alors les deuxième et troisième compilation pour s'assurer qu'il arrive à se reproduire lui-même sans fautes, ce qui semble vouloir dire qu'il a été compilé correctement.

Enfin, installez le package :

```
make install
```

En touche finale, nous créons le lien symbolique `/tools/bin/cc`. Beaucoup de programmes et de scripts lancent **cc** au lieu de **gcc**, ceci pour conserver des programmes génériques et donc utilisables sur tout type de système Unix. Tout le monde n'a pas le compilateur GNU C installé. Utiliser **cc** permet à l'administrateur système de choisir le compilateur C à installer, tant qu'un lien symbolique est créé :

```
ln -sf gcc /tools/bin/cc
```

Installer Linux-2.4.22 headers

```
Estimation du temps de construction :   0,1 SBU
Estimation de l'espace disque requis :  186 Mo
```

Contenu de Linux

Le noyau Linux kernel est au coeur de chaque système Linux. C'est ce qui fait tourner Linux. Lorsqu'un ordinateur est allumé et lance un système Linux, la première pièce de logiciel Linux à être chargé est le noyau. Celui-ci initialise les composants matériels du système: ports séries, ports parallèles, cartes son, cartes réseaux, contrôleurs IDE, contrôleurs SCSI et beaucoup plus encore. EN bref, le noyau rend le matériel disponible aux logiciels exécutés.

Fichiers installés : le noyau et ses en-têtes.

Dépendances d'installation de Linux

Linux dépend de Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Installation des en−têtes du noyau

Comme certains packages ont besoin de se référer aux fichiers d'en−tête du noyau, nous allons déballer l'archive du noyau maintenant, les configurer et copier les fichiers requis à un endroit où **gcc** pourra les trouver plus tard.

Préparez l'installation des en−têtes avec :

```
make mrproper
```

Ceci nous assure que le répertoire du noyau est totalement propre. L'équipe du noyau recommande que cette commande soit exécutée à *chaque* compilation du noyau. Vous ne devriez pas vous reposer sur un répertoire non nettoyé après le déballage.

Créez le fichier `include/linux/version.h` :

```
make include/linux/version.h
```

Créez le lien symbolique spécifique à la platforme `include/asm` :

```
make symlinks
```

Installez les fichiers d'en−têtes spécifiques à la platforme :

```
mkdir /tools/include/asm
cp include/asm/* /tools/include/asm
cp −R include/asm−generic /tools/include
```

Installez les autres fichiers d'en−têtes du noyau :

```
cp −R include/linux /tools/include
```

Il existe quelques fichiers d'en−têtes du noyau utilisant le fichier `autoconf.h`. Comme nous ne configurons pas encore le noyau, nous avons besoin de créer ce fichier nous−même pour éviter les erreurs de compilation. Créez un fichier `autoconf.h` vide :

```
touch /tools/include/linux/autoconf.h
```

Installer Glibc−2.3.2

```
Estimation du temps de construction :   11,8 SBU
Estimation de l'espace disque requis :  800 Mo
```

Contenu de Glibc

(Dernière vérification effectuée auprès de la version 2.3.2.)

Glibc est une bibliothèque C qui apporte les appels système et les fonctions de base telles que open, malloc, printf, etc. La bibliothèque C est utilisée par tous les programmes liés dynamiquement.

Programmes installés : catchsegv, gencat, getconf, getent, glibcbug, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nscd, nscd_nischeck, pcprofiledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump et zic

Bibliothèques installées : ld.so, libBrokenLocale.[a,so], libSegFault.so, libanl.[a,so], libbsd−compat.a, libc.[a,so], libc_nonshared.a, libcrypt.[a,so], libdl.[a,so], libg.a, libieee.a, libm.[a,so], libmcheck.a, libmemusage.so, libnsl.a, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.[a,so], libresolv.[a,so], librpcsvc.a, librt.[a,so], libthread_db.so et libutil.[a,so]

Dépendances d'installation de Glibc

Glibc dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed, Texinfo.

Installation de Glibc

Avant de commencer l'installation de la Glibc, vous devez entrer (**cd**) dans le répertoire `glibc−2.3.2` et déballer Glibc−linuxthreads dans ce répertoire, et non pas dans le répertoire où vous déballez habituellement tous les sources.

> **Note :** Nous allons lancer la suite de tests pour Glibc dans ce chapitre. Néanmoins, il est important de noter que lancer la suite de tests Glibc ici n'est pas considéré comme étant aussi important que dans le chapitre 6.

Ce package est connu pour mal se comporter si vous changez les options d'optimisation par défaut (en incluant les options −march et −mcpu). Donc, si vous avez défini des variables d'environnement qui surchargent les optimisations pas défaut, telles que CFLAGS et CXXFLAGS, nous vous recommandons de supprimer cette initialisation lors de la construction de Glibc.

Basiquement, compiler Glibc de tout autre façon que celle proposée par le livre compromet la stabilité de votre système.

Bien qu'il s'agisse d'un message sans gravité, l'étape d'installation de Glibc se plaindra de l'absence de `/tools/etc/ld.so.conf`. Corrigez ce petit message ennuyant avec :

```
mkdir /tools/etc
touch /tools/etc/ld.so.conf
```

De même, Glibc a un problème subtil s'il est compilé avec GCC 3.3.1. Appliquez le correctif suivant pour corriger ceci :

```
patch -Np1 -i ../glibc-2.3.2-sscanf-1.patch
```

La documentation de Glibc recommande de construire Glibc en dehors du répertoire des sources, c'est-à-dire dans un répertoire dédié :

```
mkdir ../glibc-build
cd ../glibc-build
```

Ensuite, préparez la compilation de Glibc :

```
../glibc-2.3.2/configure --prefix=/tools \
    --disable-profile --enable-add-ons \
    --with-headers=/tools/include \
    --with-binutils=/tools/bin \
    --without-gd
```

La signification des options de configure est :

- **`--disable-profile`** : Ceci désactive la construction des bibliothèques avec les informations de profilage. Enlevez cette option si vous pensez faire du profilage.
- **`--enable-add-ons`** : Ceci active tous les composants supplémentaires (*add-ons*) installés avec Glibc, dans notre cas Linuxthreads.
- **`--with-binutils=/tools/bin`** et **`--with-headers=/tools/include`** : Ces options ne sont pas réellement nécessaires. Mais elles nous assurent que rien de mauvais ne peut arriver en ce qui concerne les en-têtes du noyau et que les programmes Binutils sont utilisés lors de la construction de Glibc.
- **`--without-gd`** : Cette option nous assure que le programme **`memusagestat`** ne sera pas construit, car de façon assez étrange celui-ci insiste pour être lié avec les bibliothèques de l'hôte (libgd, libpng, libz et ainsi de suite).

Lors de cette étape, vous rencontrerez peut-être le message suivant :

```
configure: WARNING:
*** These auxiliary programs are missing or incompatible versions: msgfmt
*** some features will be disabled.
*** Check the INSTALL file for required versions.
```

Le programme `msgfmt`, manquant ou incompatible, ne pose généralement pas de problème mais certaines personnes pensent qu'il peut poser problème lors de l'exécution de la suite de tests.

Compilez le package :

```
make
```

Lancez la suite de tests :

```
make check
```

La suite de tests de Glibc est grandement dépendante de certaines fonctionnalités de votre système hôte, en particulier du noyau. De plus, dans ce chapitre, certains tests peuvent être affecter durement par les outils existants ainsi que par des problèmes d'environnement sur le système hôte. Bien sûr, ceci ne sera plus un

problème lorsque nous lancerons la suite de tests de Glibc dans l'environnement chroot au cours du chapitre 6. En général, la suite de tests de Glibc réussit. Néanmoins, comme mentionné ci−dessus, quelques échecs sont inévitables sous certaines circonstances. Voici une liste des problèmes les plus communs que nous connaissons :

- Le test *math* échoue parfois lorsqu'il est exécuté sur des systèmes où le processeur n'est pas un nouveau Intel ou AMD. Certains paramètrages d'optimisation sont aussi connus pour être la source de ce type de problèmes.
- Le test *gettext* échoue quelque fois à cause de problèmes sur le système hôte. Les raisons exactes ne sont pas encore claires.
- Le test *atime* échoue quelque fois lorsque la partition LFS est montée avec l'option *noatime* ou à cause d'autres spécificités des autres systèmes de fichiers.
- Le test *shm* pourrait échouer si le système hôte utilise le système de fichiers devfs mais ne dispose pas du système de fichiers tmpfs monté sur `/dev/shm` à cause d'un manque de support de tmpfs dans le noyau.
- En cas d'exécution sur d'anciens matériels (lents), certains tests pourraient échouer à cause de limite de temps dépassée.

En résumé, ne vous inquiétez pas trop si vous voyez des échecs lors des tests de la Glibc dans ce chapitre. La Glibc au chapitre 6 est celle avec qui nous allons rester, donc c'est celle−ci pour qui les tests doivent réussir. Mais gardez en tête, y compris au chapitre 6 que certaines erreurs peuvent arriver, les tests de *math* par exemple. Si vous essuyez un échec, notez−le puis essayez de relancer **`make check`**. La suite de tests devrait reprendre là où elle s'était arrêtée. Vous pouvez passer cette séquence arrêt−relancement en lançant un **`make −k check`**. Mais en faisant cela, assurez−vous de sauvegarder la sortie pour que vous puissiez examiner les traces plus tard et savoir ainsi le nombre total d'échecs.

Maintenant, installez le package :

```
make install
```

Différents pays et cultures ont des conventions variables sur la façon de communiquer. Ces conventions vont du très simple, telle que la représentation de la date et de l'heure à du très compliqué, telle que le langage parlé. L'internationalisation des programmes GNU fonctionne en utilisant les *locales*. Nous installons les locales Glibc maintenant :

```
make localedata/install−locales
```

Une alternative au lancement de la commande précédente est d'installer uniquement les locales dont vous avez besoin ou que vous souhaitez. Ceci est possible en utilisant la commande **`localedef`**. Des informations sur cette commande sont disponibles dans le fichier `INSTALL` des sources `glibc−2.3.2`. Néanmoins, il existe un certain nombre de locales essentielles pour réussir les tests des futurs packages, en particulier les tests de *libstdc++* provenant de GCC. Les instructions suivantes, au lieu de la cible install−locales comme ci−dessus, installent l'ensemble minimum de locales nécessaires à la bonne réussite des tests :

```
mkdir −p /tools/lib/locale
localedef −i de_DE −f ISO−8859−1 de_DE
localedef −i de_DE@euro −f ISO−8859−15 de_DE@euro
localedef −i en_HK −f ISO−8859−1 en_HK
localedef −i en_PH −f ISO−8859−1 en_PH
localedef −i en_US −f ISO−8859−1 en_US
localedef −i es_MX −f ISO−8859−1 es_MX
localedef −i fr_FR −f ISO−8859−1 fr_FR
```

```
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
```

"Verrouiller" Glibc

Maintenant que les bibliothèques C temporaires ont été installées, nous voulons que les outils compilés avec le reste de ce chapitre soient liés avec ces bibliothèques. Pour accomplir ceci, nous avons besoin d'ajuster l'éditeur de liens et le fichier specs du compilateur.

Tout d'abord, installez les scripts améliorés de l'éditeur de liens en lançant ce qui suit à l'intérieur du répertoire `binutils-build` :

```
make -C ld install
```

Ces scripts ont été améliorés un peu avant, à la fin de la première passe de Binutils et ne contiennent aucune mention de `/lib`, `/usr/lib` ou `/usr/local/lib`. A partir de ça, tout va être lier *seulement* avec les bibliothèques dans `/tools/lib`.

L'éditeur de liens a été ajusté un peu auparavant, à la fin de la première passe de Binutils. A partir de maintenant, tout sera lié *uniquement* avec les bibliothèques contenues dans `/tools/lib`.

Maintenant que l'éditeur de liens ajusté est installé, vous pouvez supprimer les répertoires des sources et de construction de Binutils.

L'autre chose à faire est de modifier le fichier specs de GCC de façon à ce qu'il pointe vers le nouvel éditeur de liens dynamiques. Une simple commande sed accomplira ceci :

```
SPECFILE=/tools/lib/gcc-lib/*/*/specs
sed -e 's@/lib/ld.so.1@/tools/lib/ld.so.1@g' \
    -e 's@/lib/ld-linux.so.2@/tools/lib/ld-linux.so.2@g' \
    $SPECFILE > tempspecfile
mv tempspecfile $SPECFILE
unset SPECFILE
```

Nous recommandons de copier/coller les commandes ci-dessus plutôt que d'essayer de taper tout ce qui s'y trouve. Ou vous pouvez éditer le fichier specs à la main si vous le voulez : remplacez "/lib/ld-linux.so.2" par "/tools/lib/ld-linux.so.2" et "/lib/ld.so.1" avec "/tools/lib/ld.so.1".

En dernier lieu, il existe une possibilité que certains fichiers include de l'hôte système se trouvent dans le répertoire include privé de GCC. Ceci peut se produire parce que la procédure fixincludes de GCC est lancée en tant que partie la construction GCC. Nous expliquerons un peu mieux ceci dans ce chapitre. Pour l'instant, lancez les commandes suivantes pour éliminer cette possibilité.

```
rm -f /tools/lib/gcc-lib/*/*/include/{pthread.h,bits/sigthread.h}
```

Attention
Il est impératif à ce moment de s'arrêter pour s'assurer que les fonctions de base (compilation et édition de liens) du nouvel ensemble des outils fonctionneront comme on s'y attend. Pour cela, nous allons effectuer une vérification simple :

```
echo 'main(){}' > dummy.c
gcc dummy.c
readelf -l a.out | grep ': /tools'
```

Si tout fonctionne correctement, la sortie de la dernière commande devrait être :

```
[Requesting program interpreter: /tools/lib/ld-linux.so.2]
```

Si vous n'obtenez pas une sortie comme celle montrée ci-dessus, alors quelque chose va très mal. Vous devrez enquêter là-dessus et reprendre chaque étape pour trouver où est situé le problème et le corriger. Il ne sert à rien de continuer jusqu'à ce que ce soit corrigé. Il est probable que quelque chose s'est mal passé avec le fichier specs ci-dessus. Vérifiez particulièrement que `/tools/lib` apparaît comme le préfixe de notre éditeur de liens dynamiques. Bien sûr, si vous travaillez sur une plateforme où le nom de l'éditeur de liens est quelque chose d'autre que `ld-linux.so.2`, alors la sortie sera un peu différente.

Lorsque vous êtes sûr que tout va bien, effacez les fichiers de test :

```
rm dummy.c a.out
```

Ceci complète l'installation de l'ensemble d'outils qui pourra maintenant être utilisée pour construire le reste des outils temporaires.

Installer Tcl-8.4.4

```
Estimation du temps de construction :   0,9 SBU
Estimation de l'espace disque requis :  23 Mo
```

Contenu de Tcl

(Dernière vérification effectuée auprès de la version 8.4.4.)

Le package Tcl contient le langage de commandes outils (Tool Command Language).

Tcl installe les fichiers suivants:

Programmes installés : tclsh (lien vers tclsh8.4), tclsh8.4

Bibliothèque installée : libtcl8.4.so

Dépendances d'installation de Tcl

Tcl dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation de Tcl

Ce package et les deux suivants sont seulement installés dans le but de lancer les suites de tests pour GCC et Binutils. Installer ces trois packages juste pour des tests peut sembler beaucoup mais il est très rassurant, voire essentiel, de savoir que nos outils les plus importants fonctionnent correctement.

Préparez la compilation de Tcl :

```
cd unix
./configure --prefix=/tools
```

Construisez le package :

```
make
```

Ce package dispose d'une suite de tests pouvant réaliser un certain nombre de vérification pour nous assurer que tout est construit correctement. Néanmoins, la suite de tests Tcl dans ce chapitre est connu pour ses échecs sous certaines conditions dépendant de l'hôte et qui ne sont pas totalement comprises. Du coup, ces échecs ne sont pas surprenant et ne doivent pas être considérés comme critiques. Si vous souhaitez lancer le suite de tests, la commande suivante le fera :

```
TZ=UTC make test
```

La signification du paramètre de make est :

- **TZ=UTC** : Ceci initialise la zone horaire à l'UTC (Coordinated Universal Time) aussi connue comme GMT (Greenwich Mean Time), mais seulement pendant la durée de cette suite de tests. Ceci nous assure que les tests concernant l'horloge seront réalisés correctement. Plus d'informations sur la variable d'environnement TZ seront disponibles plus tard sur chapitre 7.

Quelque fois, les suites de tests des packages échouent pour des raisons erronées. Vous pouvez consulter le Wiki LFS sur http://wiki.linuxfromscratch.org/ pour vérifier si ces échecs sont normaux. Ceci s'applique à tous les tests du livre.

Installez le package :

```
make install
```

> **Important :** *Ne supprimez pas* encore le répertoire source `tcl8.4.4`, car le package suivant en aura besoin pour ses en−têtes internes.

Créez le lien symbolique nécessaire :

```
ln -s tclsh8.4 /tools/bin/tclsh
```

Installer Expect−5.39.0

```
Estimation du temps de construction :   0,1 SBU
Estimation de l'espace disque requis :  3,9 Mo
```

Contenu de Expect

Le package Expect contient un programme exécutant un dialogue programmé avec d'autres programmes interactifs.

Programme installé : expect

Bibliothèque installée : libexpect5.39.a

Dépendances d'installation d'Expect

Expect dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, Tcl.

Installation of Expect

First apply a patch:

```
patch -Np1 -i ../expect-5.39.0-spawn.patch
```

This fixes a bug in Expect that can result in bogus failures during the GCC test suite run.

Now prepare Expect for compilation:

```
./configure --prefix=/tools --with-tcl=/tools/lib --with-x=no
```

The meaning of the configure options:

- **`--with-tcl=/tools/lib`**: This ensures that the configure script finds the Tcl installation in our temporary tools location. We don't want it to find an existing one that may possibly reside on the host system.
- **`--with-x=no`**: This tells the configure script not to search for Tk (the Tcl GUI component) or the X Window System libraries, both of which may possibly reside on the host system.

Build the package:

```
make
```

This package has a test suite available which can perform a number of checks to ensure it built correctly. However, the Expect test suite here in Chapter 5 is known to experience failures under certain host conditions that are not fully understood. Therefore, test suite failures here are not surprising, but are not considered critical. Should you choose to run the test suite, the following command will do so:

```
make test
```

And install:

```
make SCRIPTS="" install
```

The meaning of the make parameter:

- **`SCRIPTS=""`**: This prevents installation of the supplementary expect scripts which are not needed.

You can now remove the source directories of both Tcl and Expect.

Installer DejaGnu−1.4.3

```
Estimation du temps de construction :   0,1 SBU
Estimation de l'espace disque requis :  8,6 Mo
```

Contenu de DejaGnu

Le package DejaGnu contient un groupe de travail pour tester d'autres programmes.

Programme installé : runtest

Dépendances d'installation de DejaGnu

Dejagnu dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation de DejaGnu

Préparez la compilation de DejaGnu :

```
./configure --prefix=/tools
```

Construisez et installez le package :

```
make install
```

Installer GCC−3.3.1 − Pass 2

```
Estimation du temps de construction :   11,0 SBU
Estimation de l'espace disque requis :  274 Mo
```

Réinstallation de GCC

Les outils requis pour tester GCC et Binutils sont maintenant installés (Tcl, Expect et DejaGnu). Nous pouvons continuer en reconstruisant GCC et Binutils, en les liant avec la nouvelle Glibc et en les testant correctement. Une chose à noter, néanmoins, est que ces suites de tests dépendent énormément de pseudos terminaux (PTY) fonctionnels fournis par votre distribution hôte. Actuellement, les PTY sont souvent implémentés via le système de fichiers *devpts*. Vous pouvez rapidement vérifier si votre système hôte est correctement configuré en cela en réalisant un simple test :

```
expect -c "spawn ls"
```

Si vous obtenez le message :

```
The system has no more ptys.  Ask your system administrator to create more.
```

Votre distribution hôte n'est pas correctement configurée pour les PTY. Dans ce cas, il ne sert à rien de lancer les suites de tests de GCC et Binutils jusqu'à la correction de ce problème. Vous pouvez consulter le Wiki

LFS sur http://wiki.linuxfromscratch.org/ pour plus d'informations pour faire fonctionner les PTY.

Déballez les trois archives tar de GCC (−core, −g++ et −testsuite) dans un seul répertoire. Elles se placeront dans un seul sous−répertoire `gcc−3.3.1/`.

Tout d'abord, corrigez un problème et faites un ajustement essentiel :

```
patch −Np1 −i ../gcc−3.3.1−no_fixincludes−2.patch
patch −Np1 −i ../gcc−3.3.1−specs−2.patch
```

Le premier correctif désactive le script GCC "fixincludes". Nous l'avions mentionné brièvement mais une explication plus en détail de fixincludes est apportée ici. Sous des circonstances normales, le script GCC fixincludes parcourt votre système pour trouver les fichiers d'en−tête qui ont besoin d'être corrigés. Il pourrait trouver que certains des fichiers d'en−têtes de Glibc sur votre système devraient être corrigés, les corriger et les placer dans le répertoire des en−têtes privés de GCC. Plus tard dans le chapitre 6, après avoir installé la nouvelle Glibc, ce répertoire serait recherché avant le répertoire include du système, faisant que GCC trouverait les en−têtes corrigés du système hôte qui ne correspondraient certainement pas à la version de Glibc actuellement utilisée pour le système LFS.

Le dernier correctif modifie l'emplacement par défaut de l'éditeur de liens dynamiques de GCC (généralement `ld−linux.so.2`). il supprime aussi `/usr/include` du chemin de recherche des includes de GCC. Corriger cela maintenant plutôt qu'ajuster le fichier specs après l'installation nous assure que notre éditeur de liens dynamiques sera utilisé lors de la construction de GCC. C'est−à−dire que tous les binaires finaux (et temporaires) créés lors de la construction seront liés à la nouvelle Glibc.

> **Important :** Ces correctifs sont *critiques* pour s'assurer une construction avec succès. Ne pas oublier de les appliquer.

Créez un répertoire de construction séparé de nouveau :

```
mkdir ../gcc−build
cd ../gcc−build
```

Avant de commencer la construction de GCC, rappelez de désinitialiser toute variable d'environnement pour surcharger les options d'optimisation par défaut.

Maintenant, préparez la compilation de GCC :

```
../gcc−3.3.1/configure −−prefix=/tools \
    −−with−local−prefix=/tools \
    −−enable−clocale=gnu −−enable−shared \
    −−enable−threads=posix −−enable−__cxa_atexit \
    −−enable−languages=c,c++
```

La signification des nouvelles options de configure est :

- **`−−enable−threads=posix`** : Ceci active la gestion des exceptions C++ pour le code multi−threadé.
- **`−−enable−__cxa_atexit`** : Cette option autorise l'utilisation de __cxa_atexit, plutôt que atexit, pour enregistrer les destructeurs C++ destructors des objets statiques locales et globales et est essentiel pour la gestion des destructeurs en compatibilité complète avec les standards. Il affecte aussi l'ABI C++ et donc résulte en des bibliothèques partagées et des programmes C++ interopérables avec

les autres distributions Linux.

- **`--enable-clocale=gnu`** : Cette option nous assure que le bon modèle de locale est sélectionné pour les bibliothèques C++ sous toutes les circonstances. Si le script configure trouve la locale *de_DE* installée, il sélectionnera le bon modèle de *gnu*. Néanmoins, les personnes qui n'installent pas la locale *de_DE* courent le risque de construire les bibliothèques C++ incompatibles avec ABI dû au mauvais modèle de locale, *generic*, sélectionné.
- **`--enable-languages=c,c++`** : Cette option est nécessaire pour s'assurer que les compilateurs C et C++ sont construits.

Compilez le package :

```
make
```

Il n'est pas nécessaire d'utiliser la cible **`bootstrap`** maintenant car le compilateur que nous utilisons pour compiler ce GCC a été construit avec exactement la même version des sources de GCC utilisées précédemment.

> **Note :** Il est important de noter que lancer la suite de tests de GCC maintenant n'est pas considéré comme aussi important que de la lancer dans le chapitre 6.

Testez le résultat :

```
make -k check
```

L'option **`-k`** est utilisée pour faire en sorte que toute la suite de tests soit exécutée et ne s'arrête pas au premier échec. La suite de tests GCC est très complète et il est pratiquement garanti que certaines erreurs apparaîtront. Pour obtenir un résumé des résultats de la suite de tests, lancez ceci :

```
../gcc-3.3.1/contrib/test_summary | more
```

Vous pouvez comparer vos résultats avec ceux postés sur la liste de diffusion gcc-testresults pour des configurations similaires à la vôtre. Pour un exemple concernant la version actuelle GCC-3.3.1 sur un i686-pc-linux-gnu, voir http://gcc.gnu.org/ml/gcc-testresults/2003-08/msg01612.html.

Notez que les résultats contiennent :

```
* 1 XPASS (unexpected pass) for g++
* 1 FAIL (unexpected failure) for g++
* 2 FAIL for gcc
* 26 XPASS's for libstdc++
```

La partie échouée ("unexpected pass") pour g++ est dûe à l'utilisation de **`--enable-__cxa_atexit`**. Apparemment, toutes les plateformes supportées par GCC ne disposent pas du support de "__cxa_atexit" dans leur bibliothèques C, donc il est probable que ce test ne réussira pas.

Les 26 parties échouées pour libstdc++ sont dues à l'utilisation de **`--enable-clocale=gnu`**, qui est le choix correct pour des systèmes basées sur Glibc, versions 2.2.5 et supérieurs. Le support de la locale dans la bibliothèque GNU C est supérieurà l'autre modèle sélectionné ("generic") qui pourrait être applicable si par exemple vous utilisiez Newlibc, Sun-libc ou tout autre libc). La suite de tests libstdc++ attend apparemment le modèle "generic", donc ces tests ne passent pas obligatoirement.

Des échecs inattendues ne peuvent souvent pas être évités. Les développeurs GCC sont habituellement au courant mais n'ont pas encore réussi à les corriger. En bref, sauf si vos résultats sont grandement différents de ceux présents sur l'URL ci−dessus, il est sain de continuer.

Et finalement, installez le package :

```
make install
```

> **Note :** A ce moment, il est fortement recommandé de répéter la vérification que nous avions réalisé dans ce chapitre. Référez−vous au la section intitulée *"Verrouiller" Glibc* et répétez la vérification. Si les résultats sont mauvais, alors vous avez probablement oublié d'appliquer le correctif "GCC Specs" mentionné ci−dessus.

Installer Binutils−2.14 − Pass 2

```
Estimation du temps de construction :   1,5 SBU
Estimation de l'espace disque requis :  108 Mo
```

Ré−installation de Binutils

Créez de nouveau un répertoire de construction séparé :

```
mkdir ../binutils-build
cd ../binutils-build
```

Maintenant, préparez la compilation de Binutils :

```
../binutils-2.14/configure --prefix=/tools \
    --enable-shared --with-lib-path=/tools/lib
```

La signification des nouvelles options de configure est la suivante :

- **`--with−lib−path=/tools/lib`**: Ceci indique au script configure de spécifier le chemin de recherche par défaut des bibliothèques. Nous ne voulons pas que le chemin de recherche contienne des répertoires du système hôte.

Avant de commencer la construction de Binutils, rappellez−vous de dé−initialiser toute variable d'environnement qui surcharge les options d'optimisation par défaut.

Compilez le package :

```
make
```

> **Note :** Il est important de noter que lancer la suite de tests de Binutils ici n'est pas considéré aussi important que dans le chapitre 6.

Testez le résultat (il ne devrait pas y avoir d'échecs inattendus ici, les échecs attendus sont bons) :

```
make check
```

Malheureusement, il n'existe pas de façon facile de visualiser le résumé des résultats des tests comme c'était le cas pour le précédent package GCC. Néanmoins, si un échec arrive, il devrait être facile à trouver. La sortie devrait contenir ceci :

```
make[1]: *** [check-binutils] Error 2
```

Enfin, installez le package :

```
make install
```

Maintenant, préparez Binutils pour le ré−ajustement de l'ensemble des outils du chapitre suivant :

```
make -C ld clean
make -C ld LIB_PATH=/usr/lib:/lib
```

Avertissement
Ne supprimez pas encore les répertoires des sources et de construction de Binutils. Nous aurons de nouveau besoin de ces répertoires dans le prochain chapitre dans l'état où il est actuellement.

Installer Gawk−3.1.3

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  17 Mo
```

Contenu de Gawk

Gawk est une implémentation de awk qui est utilisée pour manipuler des fichiers texte.

Programmes installés : awk (lien vers gawk), gawk, gawk−3.1.3, grcat, igawk, pgawk, pgawk−3.1.3 et pwcat.

Dépendances d'installation de Gawk

Gawk dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Installation de Gawk

Préparez la compilation de Gawk :

```
./configure --prefix=/tools
```

Compilez le package :

```
make
```

Ce package dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Et installez le package :

```
make install
```

Installer Coreutils−5.0

```
Estimation du temps de construction :  0,9 SBU
Estimation de l'espace disque requis :  69 Mo
```

Contenu de Coreutils

Le package Coreutils contient un grand ensemble d'utilitaires shell basiques.

Programmes installés : basename, cat, chgrp, chmod, chown, chroot, cksum, comm, cp, csplit, cut, date, dd, df, dir, dircolors, dirname, du, echo, env, expand, expr, factor, false, fmt, fold, groups, head, hostid, hostname, id, install, join, kill, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mv, nice, nl, nohup, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, rm, rmdir, seq, sha1sum, shred, sleep, sort, split, stat, stty, su, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, uptime, users, vdir, wc, who, whoami et yes

Dépendances d'installation de Coreutils

Coreutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed.

Installation de Coreutils

Préparez la compilation de Coreutils :

```
./configure --prefix=/tools
```

Compilez le package :

```
make
```

Ce package dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make RUN_EXPENSIVE_TESTS=yes check
```

La signification du paramètre de make est :

- **`RUN_EXPENSIVE_TESTS=yes`** : Ceci indique à la suite de tests de lancer plusieurs tests supplémentaires considérés habituellement longs sur certaines plate−formes. Néanmoins, elles ne sont généralement pas un problème sous Linux.

Et installez le package :

```
make install
```

Installer Bzip2−1.0.2

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  2,5 Mo
```

Contenu de Bzip2

Bzip2 est un compresseur de fichiers qui arrive habituellement à une meilleure compression que ne le fait **`gzip`** traditionel.

Programmes installés : bunzip2 (lien vers bzip2), bzcat (lien vers bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless et bzmore

Bibliothèques installées : libbz2.a, libbz2.so (lien vers libbz2.so.1.0), libbz2.so.1.0 (lien vers libbz2.so.1.0.2) et libbz2.so.1.0.2

Dépendances d'installation de Bzip2

Bzip2 dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make.

Installation de Bzip2

Le package Bzip2 ne contient pas de script **`configure`**. Compilez et installez−le ainsi :

```
make PREFIX=/tools install
```

Installer Gzip−1.3.5

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  2,6 Mo
```

Contenu de Gzip

Le package Gzip contient des programmes pour compresser et décompresser des fichiers utilisant le codage Lempel−Ziv (LZ77).

Programmes installés : gunzip (lien vers gzip), gzexe, gzip, uncompress (lien vers gunzip), zcat (lien vers gzip), zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore et znew

Dépendances d'installation de Gzip

Gzip dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation de Gzip

Préparez la compilation de Gzip :

```
./configure --prefix=/tools
```

Compilez le package :

```
make
```

Et installez−le :

```
make install
```

Installer Diffutils−2.8.1

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  7,5 Mo
```

Contenu de Diffutils

Les programmes de ce package vous montre les différences entre les deux fichiers ou répertoires. Son utilisation la plus commune est la création de correctifs.

Programmes installés : cmp, diff, diff3 et sdiff

Dépendances d'installation de Diffutils

Diffutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Installation de Diffutils

Préparez la compilation de Diffutils :

```
./configure --prefix=/tools
```

Compilez le package :

```
make
```

Et installez−le :

```
make install
```

Installer Findutils−4.1.20

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  7,6 Mo
```

Contenu de Findutils

Le package Findutils contient les programmes de recherche de fichiers, soit en direct (en faisant une recherche récursive en direct à travers les répertoires et seulement en affichant des fichiers qui remplissent ses spécifications) soit en cherchant à travers une base de données.

Programmes installés : bigram, code, find, frcode, locate, updatedb et xargs

Dépendances d'installation de Findutils

Findutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Installation de Findutils

Préparez la compilation de Findutils :

```
./configure --prefix=/tools
```

Compilez le package :

```
make
```

Ce package dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Et installez le package :

```
make install
```

Installer Make−3.80

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  8,8 Mo
```

Contenu de Make

Make détermine, automatiquement, quelles pièces d'un grand programme ont besoin d'être recompilées et envoie les commandes pour les recompiler.

Programme installé : make

Dépendances d'installation de Make

Make dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Sed.

Installation de Make

Préparez la compilation de Make :

```
./configure --prefix=/tools
```

Compilez le programme :

```
make
```

Ce package dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Puis installez le package et sa documentation :

```
make install
```

Installer Grep-2.5.1

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  5,8 Mo
```

Contenu de Grep

Grep est un programme utilisé pour imprimer des lignes d'un fichier ressemblant à un modèle spécifié.

Programmes installés : egrep (lien vers grep), fgrep (lien vers grep) et grep

Dépendances d'installation de Grep

Grep dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Sed, Texinfo.

Installation de Grep

Préparez la compilation de Grep :

```
./configure --prefix=/tools \
    --disable-perl-regexp --with-included-regex
```

La signification des options de configure est :

- **`--disable-perl-regexp`** : Ceci nous assure que **`grep`** ne sera pas lié avec une bibliothèque PCRE qui pourrait être présente sur l'hôte mais ne serait pas disponible une fois entré dans l'environnement chroot.
- **`--with-included-regex`** : Ceci nous assure que Grep utilise son code interne pour les expressions rationnelles. Sans cela, il utiliserait le code de Glibc, connu pour être légèrement bogué.

Compilez les programmes :

```
make
```

Ce package dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Puis, installez−les ainsi que leur documentation :

```
make install
```

Installer Sed−4.0.7

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  5,2 Mo
```

Contenu de Sed

sed est un éditeur en flux. Un éditeur en flux est utilisé pour réaliser des transformations de texte basique sur une entrée en flux (un fichier ou une entrée à partir d'un tuyau).

Programme installé : sed

Dépendances d'installation de Sed

(Dernière vérification effectuée auprès de la version 3.02.)

Sed dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Texinfo.

Installation de Sed

Préparez la compilation de Sed :

```
./configure --prefix=/tools
```

Compilez le programme :

```
make
```

Ce package dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Puis, installez−le avec sa documentation :

```
make install
```

Installer Gettext−0.12.1

```
Estimation du temps de construction :   7,2 SBU
Estimation de l'espace disque requis :  55 Mo
```

Contenu de Gettext

Le package Gettext est utilisé pour l'internationalisation et la localisation. Les programmes peuvent êtres compilés avec le support de la langue native ('Native Language Support' ou NLS) qui leur permettent d'afficher les messages dans la langue native de l'utilisateur.

Programmes installés : autopoint, config.charset, config.rpath, gettext, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, project−id, team−address, trigger, urlget, user−email et xgettext

Bibliothèques installées : libasprintf[a,so], libgettextlib[a,so], libgettextpo[a,so] et libgettextsrc[a,so]

Dépendances d'installation de Gettext

Gettext dépend de Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Installation de Gettext

Préparez la compilation de Gettext :

```
./configure --prefix=/tools
```

Compilez les programmes :

```
make
```

Ce package dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Néanmoins, la suite de tests de Gettext est connu pour ses échecs lors du chapitre 5 sous certaines conditions dues à l'hôte, par exemple s'il trouve un compilateur Java sur l'hôte. La suite de tests Gettex prend un assez long moment à s'exécuter et n'est pas considérée comme étant critique. Donc, nous ne vous recommendons pas de la lancer maintenant. Si vous décidez de le faire, la commande suivante fera le nécessaire :

```
make check
```

Et installez le package :

```
make install
```

Installer Ncurses−5.3

```
Estimation du temps de construction :   0,7 SBU
Estimation de l'espace disque requis :  26 Mo
```

Contenu de Ncurses

(Dernière version effectuée auprès de la version 5.3.)

Le package Ncurses apporte des bibliothèques de gestion de caractères et de terminaux, incluant les panneaux et les menus.

Programmes installés : captoinfo (lien vers tic), clear, infocmp, infotocap (lien vers tic), reset (lien vers tset), tack, tic, toe, tput et tset

Bibliothèques installées : libcurses.[a,so] (lien vers libncurses.[a,so]), libform.[a,so], libmenu.[a,so], libncurses++.a, libncurses.[a,so], libpanel.[a,so]

Dépendances d'installation de Ncurses

Ncurses dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Installation de Ncurses

Corrigez deux petits problèmes :

```
patch −Np1 −i ../ncurses−5.3−etip−2.patch
patch −Np1 −i ../ncurses−5.3−vsscanf.patch
```

Le premier correctif prend en compte le fichier d'en−tête `etip.h` alors que le second empêche quelques avertissements du compilateur à cause des en−têtes obsolètes.

Maintenant, préparez la compilation de Ncurses :

```
./configure −−prefix=/tools −−with−shared \
    −−without−debug −−without−ada −−enable−overwrite
```

La signification des options de configure est :

- **`−−without−ada`** : Ceci indique à Ncurses de ne pas construire ses liens avec Ada, y compris si un compilateur Ada est installé sur l'hôte. Ceci est nécessaire car, lorsque nous entrerons dans l'environnement chroot, Ada ne sera plus disponible.
- **`−−enable−overwrite`** : Ceci indique à Ncurses d'installer les fichiers d'en−tête dans `/tools/include` au lieu de `/tools/include/ncurses` pour s'assurer que les autres packages trouveront bien les en−têtes de Ncurses.

Compilez les programmes et les bibliothèques :

```
make
```

Puis, installez−les avec leur documentation :

```
make install
```

Installer Patch−2.5.4

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  1,9 Mo
```

Contenu de Patch

Le programme patch modifie un fichier suivant un fichier patch, appelé aussi correctif. Un correctif est habituellement une liste, créée par le programme diff, et contenant les instructions sur la façon dont le fichier original a été modifié.

Programme installé : patch

Dépendances d'installation de Patch

Patch dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation de Patch

Préparez la compilation de Patch :

```
CPPFLAGS=−D_GNU_SOURCE ./configure −−prefix=/tools
```

L'option du préprocesseur **−D_GNU_SOURCE** est nécessaire uniquement pour les plateformes PowerPC. Vous pouvez l'oublier pour les autres architectures.

Compilez le programme :

```
make
```

Puis, installez−la avec sa documentation :

```
make install
```

Installer Tar−1.13.25

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  10 Mo
```

Contenu de Tar

Tar est un programme d'archivage permettant de stocker et d'extraire des fichiers à partir d'un fichier d'archive connu comme un fichier tar.

Programmes installés : rmt et tar

Dépendances d'installation de Tar

Tar dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Installation de Tar

Préparez la compilation de Tar :

```
./configure --prefix=/tools
```

Compilez les programmes :

```
make
```

Ce package dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Puis, installez−les avec leur documentation :

```
make install
```

Installer Texinfo−4.6

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  16 Mo
```

Contenu de Texinfo

Le package Texinfo contient des programmes utilisés pour lire, écrire et convertir des documents Info, qui apporte une documentation système.

Programmes installés : info, infokey, install−info, makeinfo, texi2dvi et texindex

Dépendances d'installation de Texinfo

Texinfo dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed.

Installation de Texinfo

Préparez la compilation de Texinfo :

```
./configure --prefix=/tools
```

Compilez les programmes :

```
make
```

Ce package dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Puis, installez−les avec leur documentation :

```
make install
```

Installer Bash−2.05b

```
Estimation du temps de construction :           1,2 SBU
Estimation de l'espace disque requis :  27 Mo
```

Contenu de Bash

Bash, acronyme de Bourne−Again SHell, est un interpréteur de commande très répandu sur les systèmes Unix. Il se charge d'interpréter et d'évaluer les commandes que l'utilisateur entre sur l'entrée standard (le clavier), comme par exemple lancer un programme.

Programmes installés : bash, sh (lien vers bash) et bashbug

Dépendances d'installation de Bash

Bash dépend de Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Sed.

Installation de Bash

Bash contient un certain nombre de bogues connus. Corrigez−les avec les correctifs suivants :

```
patch −Np1 −i ../bash−2.05b−2.patch
```

Maintenant, préparez la compilation de Bash :

```
./configure −−prefix=/tools
```

Compilez le programme :

```
make
```

Ce package dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make tests
```

Puis, installez−le ainsi que sa documentation :

```
make install
```

Ajoutez un lien pour les programmes utilisant **sh** comme shell :

```
ln -s bash /tools/bin/sh
```

Installer Util-linux-2.12

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  8 Mo
```

Contenu de Util-linux

Le package Util-linux contient un certain nombre d'utilitaires divers. Certains des plus connus d'entre eux sont utilisés pour monter, démonter, formatter, partionner et gérer des disques durs, ouvrir des ports tty ainsi que récupérer des messages du noyau.

Programmes installés : agetty, arch, blockdev, cal, cfdisk, chkdupexe, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, dmesg, elvtune, fdformat, fdisk, fsck.cramfs, fsck.minix, getopt, hexdump, hwclock, ipcrm, ipcs, isosize, kill, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, parse.bash, parse.tcsh, pg, pivot_root, ramsize (lien vers rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (lien vers rdev), script, setfdprm, setsid, setterm, sfdisk, swapoff (lien vers swapon), swapon, test.bash, test.tcsh, tunelp, ul, umount, vidmode (lien vers rdev), whereis et write

Dépendances d'installation de Util-linux

Util-linux dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Zlib.

Installation de Util-linux

Util-linux n'utilise pas l'installation toute fraîche des headers et librairies provenant du répertoire /tools. Ceci est résolu en modifiant le script configure :

```
cp configure configure.backup
sed "s@/usr/include@/tools/include@g" configure.backup > configure
```

Préparez la compilation de Util-linux :

```
./configure
```

Compilez quelques routines d'aide :

```
make -C lib
```

Et, comme vous allez avoir besoin de quelques utilitaires contenus dans ce package, construisez uniquement ceux-ci :

```
make −C mount  mount umount
make −C text−utils  more
```

Maintenant, copiez ces programmes dans le répertoire des outils temporaires: :

```
cp mount/{,u}mount text−utils/more /tools/bin
```

Installer Perl−5.8.0

```
Estimation du temps de construction :   2,9 SBU
Estimation de l'espace disque requis :  143 Mo
```

Contenu de Perl

Le package Perl contient perl, l'acronyme de Practical Extraction and Report Language (Langage pratique d'extraction et de rapport). Perl combine certaines des meilleurs fonctionnalités des langages C, sed, awk et sh en un seul langage extrêmement puissant.

Programmes installés : a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (lien vers perl), perlbug, perlcc, perldoc, perlivp, piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, psed (lien vers s2p), pstruct (lien vers c2ph), s2p, splain and xsubpp

Bibliothèques installées : (trop nombreux pour les nommer toutes)

Dépendances d'installation de Perl

Perl dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Installation de Perl

Tout d'abord, adaptez quelques chemins en dur de la bibliothèque C :

```
patch −Np1 −i ../perl−5.8.0−libc−3.patch
```

Et assurez−vous que les extensions statiques sont bien construites :

```
chmod u+w hints/linux.sh
echo 'static_ext="IO re Fcntl"' >> hints/linux.sh
```

Maintenant, préparez la compilation de Perl :

```
./configure.gnu −−prefix=/tools
```

Compilez seulement les outils requis :

```
make perl utilities
```

Puis, copiez ces outils et leur bibliothèques :

```
cp perl pod/pod2man /tools/bin
mkdir -p /tools/lib/perl5/5.8.0
cp -R lib/* /tools/lib/perl5/5.8.0
```

Nettoyer

Les étapes de cette section sont optionnelles. Si votre partition LFS est assez petite, vous serez content d'apprendre que vous pouvez jeter certaines choses inutiles. Les exécutables et bibliothèques que vous avez construit jusqu'à maintenant contiennent à peu près 130 Mo de symboles de débogage inutiles. Supprimez ces symboles de cette façon :

```
strip --strip-unneeded /tools/{,s}bin/*
strip --strip-debug /tools/lib/*
```

La première de ces commandes supprimera environ 20 fichiers, en indiquant qu'elle ne reconnaît pas leur format de fichier. La plupart sont des scripts et non des binaires.

Faites attention à ne *pas* utiliser **`--strip-unneeded`** sur ces bibliothèques −− elles seraient détruites et vous n'auriez plus qu'à reconstruire complètement Glibc.

Pour sauver encore quelques mégaoctets, vous pouvez supprimer la documentation :

```
rm -rf /tools/{,share/}{doc,info,man}
```

Vous aurez maintenant besoin d'avoir au moins 850 Mo d'espace libre sur le système de fichiers LFS pour pouvoir construire et installer la Glibc lors de la prochaine phase. Si vous pouvez construire et installer Glibc, vous pourrez aussi construire et installer le reste.

III. Troisième Partie − Construire le système LFS

Table des matières

Chapitre 6. Installation des logiciels du système de base

Introduction

Dans ce chapitre, nous entrons dans le site de construction et démarrons la construction de notre système LFS. C'est−à−dire, nous entrons avec chroot dans le mini système Linux temporaire, et lançons l'installation de tous les paquets un par un.

L'installation de tous les logiciels est plutôt simple et vous allez probablement penser qu'il serait beaucoup plus rapide de donner les instructions génériques d'installation pour un paquet et de seulement expliquer en détail l'installation des paquets nécessitant une méthode alternative. Même si nous sommes d'accord avec ceci, nous avons choisi de donner les instructions complètes pour chaque paquet, simplement pour éviter toute confusion ou erreur.

Si vous comptez utiliser une optimisation de compilation pour les packages installés dans ce chapitre, jetez un oeil aux astuces d'optimisation dans http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt. Ces options peuvent faire qu'un programme s'exécute légèrement plus vite, mais peuvent également causer des problèmes de compilation et même des erreurs à l'exécution. Si vous rencontrez des problèmes après avoir utilisé les optimisations, essayez toujours sans optimisation pour voir si le problème persiste. Même si le package compile en utilisant les optimisations, il y a des risques pour qu'il ait été compilé de façon incorrecte à cause des interactions complexes entre le code et les outils de construction. En bref, les petits gains potentiels obtenus avec l'optimisation du compilateur sont généralement bien moindres par rapport au risque. Pour une première installation de LFS, nous vous encourageons à construire sans optimisation. Votre système sera toujours très rapide et très stable en même temps.

L'ordre dans lequel ces paquets sont installés dans ce chapitre doit être suivi à la lettre, pour s'assurer qu'aucun programme n'obtienne un chemin codé en dur se référant à `/tools`. Pour la même raison, *ne pas* compiler les paquets en parallèle. Compiler en parallèle peut vous sauver du temps (et spécialement sur les machines bi−processeur), mais cela peut avoir pour résultat un programme contenant un chemin codé en dur menant vers `/tools`, ce qui empêchera le programme de fonctionner normalement lorsque ce répertoire sera supprimé.

A propos des symboles de débogage

La plupart des programmes et des bibliothèques sont compilés, par défaut, avec les informations de débogage (avec l'option −g de gcc).

Lors du débogage d'un programme ou d'une librairie qui a été compilée avec les options de débogage, le débogueur vous donnera non seulement les adresses mémoires, mais aussi les noms des routines et des variables.

Mais l'inclusion de ces informations de débogage grossit le programme ou la bibliothèque de manière significative. Pour avoir une idée de la quantité d'espace que ces symboles occupent, regarder ceci :

- Un binaire Bash avec inforamtions de débogage : 1200 Ko.
- Un binaire Bash sans informations de débogage : 478 Ko.
- Fichiers Glibc et GCC (/lib et /usr/lib) avec informations de débogage : 87 Mo.

- Fichiers Glibc et GCC sans informations de débogage : 16 Mo.

Les tailles peuvent varier un peu selon le compilateur utilisé et la version de la bibliothèque C utilisée. Mais lors d'une comparaison entre des programmes avec les informations de débogage et ceux sans, la différence sera généralement d'un facteur 2 à 5.

Comme la plupart des gens n'utiliseront probablement jamais un débogueur sur leurs exécutable systeme, beaucoup de place disque peut être gagnée en supprimant ces informations.

Pour enlever les informations de débogage d'un binaire (qui doit être de type a.out ou ELF), exécutez **`strip --strip-debug filename`**. Les jokers peuvent être utilisés pour traiter plusieurs fichiers (utilisez quelque chose comme **`strip --strip-debug $LFS/tools/bin/*`**).

Pour vous faciliter la tâche, le chapitre 9 comprend une commande unique pour supprimer tous les symboles de débogage des programmes et bibliothèques de votre système. des informations supplémentaires sont disponibles dans l'astuce http://www.linuxfromscratch.org/hints/downloads/files/optimization.txt.

Entrée dans l'environnement chroot

C'est le moment d'entrer dans l'environnement chroot afin d'installer le reste des programmes nécessaires. Avant de lancer la commande chroot, vous devez être *root* car seul cet utilisateur peut utiliser cette commande.

Comme précédemment, assurez-vous que la variable d'environnement LFS est configurée correctement en lançant **`echo $LFS`** et en vérifiant que le chemin affiché pointe bien vers le point de montage de la partition LFS, c'est-à-dire `/mnt/lfs` si vous avez suivi notre exemple.

Devenez *root* et lancez la commande suivante pour entrer dans l'environnement chroot :

```
chroot $LFS /tools/bin/env -i \
    HOME=/root TERM=$TERM PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin:/tools/bin \
    LDFLAGS="-s" \
    /tools/bin/bash --login
```

L'option **`-i`** donné à la commande **`env`** efface toutes les variables de l'environnement chroot. Après cela, seules les variables HOME, TERM, PS1 et PATH sont initialisées de nouveau. La commande TERM=$TERM initialise la variable TERM à l'intérieur de chroot avec la même valeur que celle qu'elle avait à l'extérieur ; cette variable est utilisée par des programmes comme **`vim`** et **`less`** pour fonctionner correctement. Si vous avez besoin d'autres variables, telles que CFLAGS ou CXXFLAGS, vous pouvez aussi les initialiser ici.

Dès maintenant, nous n'avons plus besoin d'utiliser la variable LFS -- car ce que le shell pense être `/` est la valeur de `$LFS`, qui a été passée à la commande chroot.

Notez que `/tools/bin` est le dernier dans PATH. Ceci signifie qu'un outil temporaire ne sera pas utilisé tant que sa version finale ne sera pas installée. Enfin, au moins lorsque le shell ne se rappelle pas de l'emplacement des binaires exécutés, et c'est pour cette raison que le hachage est désactivé un peu plus tard.

Assurez-vous que les commandes dans le reste de ce chapitre et dans les suivants sont exécutées dans l'environnement chroot. Si vous quittez l'environnement chroot (en redémarrant l'ordinateur par exemple),

Pensez à retourner de nouveau dans l'environnement chroot et à monter une nouvelle fois les systèmes de fichiers proc et devpts (comme indiqué plus tard) avant de continuer dans le livre.

Notez que l'invite de bash contiendra "I have no name!" (je n'ai pas de nom!) ce qui est normal puisque `/etc/passwd` n'a pas encore été créé.

Changer de propriétaire

Actuellement, le répertoire `/tools` appartient à l'utilisateur *lfs*. Néanmoins, ce compte utilisateur existe seulement sur votre système hôte. Bien que vous puissiez supprimer le répertoire `/tools` une fois que vous avez terminé votre système LFS, vous pourriez vouloir le conserver, par exemple pour construire d'autres systèmes LFS. Mais si vous conservez ce répertoire tel qu'il est, vous finirez avec des fichiers appartenant à un identifiant sans compte correspondant. Ceci est dangereux car par la suite un compte utilisateur pourrait obtenir cet identifiant et devenir soudainement le propriétaire du répertoire `/tools` et les fichiers qu'il contient, les exposant à une manipulation détournée possible.

Pour éviter ce problème, vous pouvez ajouter l'utilisateur *lfs* dans votre nouveau système LFS en créant plus tard le fichier `/etc/passwd`, et en prenant garde d'affecter le bon identifiant utilisateur et groupe. Sinon, vous pouvez (et le livre va dans ce sens) d'ores et déjà affecter le contenu du répertoire `/tools` à l'utilisateur *root* :

```
chown -R 0:0 /tools
```

La commande utilise "0:0" au lieu de "root:root", car **chown** n'est pas capable de résoudre le nom "root" tant que le fichier passwd n'est pas créé.

Création des répertoires

Créons maintenant la hiérarchie de répertoires sur notre système de fichiers LFS. Lancer les commandes suivantes pour créer une hiérarchie de répertoires plus ou moins standard :

```
mkdir -p /{bin,boot,dev/{pts,shm},etc/opt,home,lib,mnt,proc}
mkdir -p /{root,sbin,tmp,usr/local,var,opt}
for dirname in /usr /usr/local
    do
    mkdir $dirname/{bin,etc,include,lib,sbin,share,src}
    ln -s share/{man,doc,info} $dirname
    mkdir $dirname/share/{dict,doc,info,locale,man}
    mkdir $dirname/share/{nls,misc,terminfo,zoneinfo}
    mkdir $dirname/share/man/man{1,2,3,4,5,6,7,8}
done &&
mkdir /var/{lock,log,mail,run,spool}
mkdir -p /var/{tmp,opt,cache,lib/misc,local}
mkdir /opt/{bin,doc,include,info}
mkdir -p /opt/{lib,man/man{1,2,3,4,5,6,7,8}}
```

Par défaut, les répertoires sont créés avec le mode de permission 755, ce qui n'est pas souhaitable pour tous les répertoires. Nous allons réaliser deux changements : un pour le répertoire principal de *root* et un autre pour les répertoires des fichiers temporaires.

```
chmod 0750 /root
chmod 1777 /tmp /var/tmp
```

Le premier changement de permission nous assure que n'importe qui ne pourra pas entrer dans le répertoire `/root` (de façon identique à un utilisateur pour son répertoire principal). Le deuxième changement assure que tout utilisateur peut écrire dans les répertoires `/tmp` et `/var/tmp`, mais ne peut pas supprimer les fichiers des autres utilisateurs. Cette dernière interdiction est due au bit dit "sticky" −− le bit le plus haut dans le masque 1777.

Remarques à propos de la conformité FHS

Nous avons basé notre arborescence de répertoires sur le standard FHS (disponible sur http://www.pathname.com/fhs/). En plus de cette arborescence, ce standard stipule l'existence de `/usr/local/games` et `/usr/share/games`, mais nous ne voyons pas l'intérêt de ceux−ci pour un système de base. Néanmoins, vous êtes libres de rendre votre système compatible FHS. En ce qui concerne la structure du sous−répertoire `/usr/local/share`, le FHS est inprécis, donc nous créons ici les répertoires qui nous semblent nécessaires.

Monter le système de fichier proc et devfs

Pour que certains programmes fonctionnent correctement, les systèmes de fichiers *proc* et *devpts* doivent être disponibles depuis l'environnement chrooté. Un système de fichiers peut être monté autant de fois et à autant d'emplacements différents que vous le souhaitez. Donc, ce n'est pas un problème que ces systèmes de fichiers proc soient déjà montés sur votre système hôte −− surtout parce que ce sont des systèmes de fichiers virtuels.

Le système de fichiers *proc* est le pseudo système de fichiers d'informations sur les processus que le noyau utilise pour donner des informations sur l'état du système.

Le système de fichiers proc est monté sous `/proc` en lançant la commande suivante :

```
mount proc /proc −t proc
```

Vous pouvez obtenir des messages d'avertissement de la commande mount, tels que :

```
warning: can't open /etc/fstab: No such file or directory
not enough memory
```

Ignorez−les, ils sont dûs au fait que le système n'est pas encore installé complètement et que quelques fichiers manquent. Le montage lui−même sera fait avec succès et c'est tout ce dont nous avons besoin à ce moment.

Le système de fichiers *devpts* a été mentionné plus tôt et est maintenant la façon la plus commune d'implémenter les pseudos−terminaux(PTY).

Le système de fichiers devpts est monté sur `/dev/pts` en lançant :

```
mount devpts /dev/pts −t devpts
```

Si cette commande devait échouer avec une erreur du genre :

```
filesystem devpts not supported by kernel
```

La cause probable de ceci est que le noyau de votre système hôte a été compilé sans le support du système de fichiers devpts. Vous pouvez vérifier quels sont systèmes de fichiers que votre noyau supporte en fouillant dans ses entrailles avec une commande telle que **`cat /proc/filesystems`**. Si un type de système de fichiers nommé *devfs* est affiché ici, alors nous serons capable de contourner le problème en montant le système de fichiers devfs de l'hôte au−dessus de la nouvelle structure `/dev` que nous créerons plus tard dans la section "Créer des périphériques (Makedev)". Si devfs n'est pas affiché, ne vous inquiétez pas car il existe une troisième façon d'obtenir des PTY fonctionnels à l'intérieur de l'environnement chroot. Nous en parlerons assez rapidement dans la section Makedev déjà mentionnée.

Rappelez−vous, si pour quelque raison que ce soit, vous arrêtez de travailler sur votre LFS et que vous recommenciez plus tard, il est important de vérifier que ces systèmes de fichiers sont toujours montés dans l'environnement chroot. Sinon, certains programmes pourraient ne pas être compilés correctement.

Création les liens symboliques essentiels

Certains programmes stockent en dur des chemins vers des programmes qui n'existent pas encore. Pour satisfaire ces programmes, nous créons un certain nombre de liens symboliques qui seront remplacés par les vrais fichiers tout au long de ce chapitre lorsque nous installerons tous les logiciels.

```
ln −s /tools/bin/{bash,cat,pwd,stty} /bin
ln −s /tools/bin/perl /usr/bin
ln −s /tools/lib/libgcc_s.so.1 /usr/lib
ln −s bash /bin/sh
```

Créer les fichiers passwd et group

Pour que *root* puisse se connecter et pour que le nom "root" soit reconnu, il doit exister les entrées adéquates dans les fichiers `/etc/passwd` et `/etc/group`.

Créez le fichier `/etc/passwd` en lançant la commande suivante :

```
cat > /etc/passwd << "EOF"
root:x:0:0:root:/root:/bin/bash
EOF
```

Le mot de passe actuel pour *root* (le "x" ici sert juste à remplir la case) sera initialisé plus tard.

Créez le fichier `/etc/group` en lançant la commande suivante :

```
cat > /etc/group << "EOF"
root:x:0:
bin:x:1:
sys:x:2:
kmem:x:3:
tty:x:4:
tape:x:5:
daemon:x:6:
floppy:x:7:
disk:x:8:
lp:x:9:
dialout:x:10:
audio:x:11:
EOF
```

Les groupes créés ne font partie d'aucun standard −− ce sont les groupes que le script MAKEDEV utilise dans la section suivante. En plus du groupe "root", le LSB (http://www.linuxbase.org) recommande seulement un groupe "bin", avec un GID de 1. Tous les autres noms de groupe et GIDs peuvent être choisis librement par l'utilisateur, car les packages bien écrits ne dépendent pas du numéro GID mais utilisent le nom du groupe.

Enfin, nous nous reconnectons dans l'environnement chroot. La résolution des noms d'utilisateurs et des noms de groupes commencera à fonctionner immédiatement après la création des fichiers `/etc/passwd` et `/etc/group`, parce que nous avons installé une Glibc complète au chapitre 5. Ceci supprime l'invite << I have no name! >>.

```
exec /tools/bin/bash --login +h
```

Notez l'utilisation de la directive **+h**. Ceci indique à **bash** de ne pas utiliser son hachage interne du chemin. Sans cette directive, **bash** se rappelerait les chemins des binaires qu'il a exécuté. Comme nous voulons utiliser nos binaires nouvellement compilés aussitôt après leur installation, nous désactivons cette fonctionnalité pour toute la durée de ce chapitre.

Créer les périphériques (Makedev−1.7)

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  50 Ko
```

Contenu de MAKEDEV

Le script MAKEDEV crée les noeuds de périphériques statiques qui résident habituellement dans le répertoire `/dev`. Des informations détaillées sur les noeuds périphériques puissent être trouvées dans le fichier `Documentation/devices.txt` dans le répertoire des sources du noyau Linux.

Script installé : MAKEDEV

Dépendances d'installation de MAKEDEV

Make dépend de Bash, Coreutils.

Créer des périphériques

Notez que déballer le fichier `MAKEDEV−1.7.bz2` ne crée pas un répertoire car le fichier ne contient qu'un script shell.

Installez le script **MAKEDEV** :

```
bzcat MAKEDEV-1.7.bz2 > /dev/MAKEDEV
chmod 754 /dev/MAKEDEV
```

Lancez le script pour créer les fichiers de périphériques :

```
cd /dev
./MAKEDEV -v generic-nopty
```

La signification des arguments est la suivante :

- **`-v`** : Ceci indique au script de se lancer en mode verbeux.
- **`generic-nopty`** : Ceci indique à **`MAKEDEV`** de créer une sélection générique des fichiers spéciaux (périphériques) utilisés habituellement, sauf pour les fichiers ptyXX et ttyXX. Nous n'avons pas besoin de ces fichiers parce que nous allons utiliser les PTY Unix98 via le système de fichiers *devpts*.

S'il s'avère que certains périphériques spéciaux `zzz` dont vous avez besoin manquent, essayez de lancer **`./MAKEDEV -v zzz`**. Autrement, vous pourriez créer des périphériques via le programme **`mknod`**. Merci de vous référer aux pages man et info si vous avez besoin de plus d'informations.

Sinon, si vous êtes incapable de monter le système de fichiers devpts plus tôt dans la section "Monter les systèmes fichiers proc et devpts", c'est le bon moment pour essayer les alternatives. Si votre noyau supporte le système de fichiers devfs, lancez la commande suivante pour monter devfs :

```
mount -t devfs devfs /dev
```

Ceci montera le système de fichiers devfs au-dessus de la nouvelle structure statique `/dev`. Ceci ne pose pas de problèmes, car les noeuds du périphériques créés sont toujours présents, ils sont juste cachés par le nouveau système de fichiers devfs.

Si ceci ne fonctionne toujours pas, la seule option laissée est d'utiliser le script MAKEDEV pour créer les fichiers ptyXX et ttyXX qui ne seraient autrement pas nécessaire. Assurez-vous de bien être dans le répertoire `/dev` puis lancez **`./MAKEDEV -v pty`**. Le mauvais côté de ceci est le suivant : nous allons créer 512 fichiers spéciaux supplémentaires qui ne seront pas nécessaire lorsque nous démarrerons enfin notre système LFS terminé.

Installer des entêtes de Linux-2.4.22

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  186 Mo
```

Contenu de Linux

Le noyau Linux kernel est au coeur de chaque système Linux. C'est ce qui fait tourner Linux. Lorsqu'un ordinateur est allumé et lance un système Linux, la première pièce de logiciel Linux à être chargé est le noyau. Celui-ci initialise les composants matériels du système: ports séries, ports parallèles, cartes son, cartes réseaux, contrôleurs IDE, contrôleurs SCSI et beaucoup plus encore. EN bref, le noyau rend le matériel disponible aux logiciels exécutés.

Fichiers installés : le noyau et ses en-têtes.

Dépendances d'installation de Linux

Linux dépend de Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Installation des en−têtes du noyau

Nous n'allons pas encore compiler un nouveau noyau −− nous le ferons quand nous aurons terminé l'installation de tous les paquets. Mais comme certains paquets ont besoin des fichiers d'en−têtes, nous allons déballer l'archive du noyau maintenant, le configurer et copier les fichiers d'en−têtes de façon à ce qu'ils soient disponibles pour ces paquets.

Il est important de noter que les fichiers du répertoire source du noyau n'appartiennent pas à *root*. Quand vous déballez un package en tant qu'utilisateur *root* (comme nous le faisons ici dans le chroot), les fichiers ont l'identifiant de l'utilisateur et du groupe du mainteneur du package sur son propre ordinateur. Ce n'est pas habituellement un problème pour tous les autres paquets que vous installez parce que vous supprimez le répertoire des sources après l'installation. Mais le répertoire des sources du noyau est souvent conservé un long moment, donc il existe une chance pour que l'identifiant utilisateur du mainteneur soit affecté à un utilisateur sur votre machine. Du coup, cette personne aurait le droit d'écriture sur les sources du noyau.

A la lumière de ceci, il est préférable de lancer **`chown -R 0:0`** sur le répertoire `linux−2.4.22` pour vous assurer que tous les fichiers appartiennent bien à l'utilisateur *root*.

Préparez l'installation des en−têtes :

```
make mrproper
```

Ceci nous assure que le répertoire des sources du noyau est totalement propre. L'équipe du noyau recommande que cette commande soit exécutée avant *chaque* compilation du noyau. Vous ne devez pas vous reposer sur un répertoire non nettoyé après son déballage.

Créez le fichier `include/linux/version.h` :

```
make include/linux/version.h
```

Créez le lien symbolique `include/asm` spécifique à la plateforme :

```
make symlinks
```

Installez les fichiers d'en−têtes spécifiques à la plateforme :

```
cp -HR include/asm /usr/include
cp -R include/asm-generic /usr/include
```

Installez les fichiers d'en−têtes inter−plate−forme :

```
cp -R include/linux /usr/include
```

Il existe quelques fichiers d'en−têtes du noyau utilisant le fichier d'en−tête `autoconf.h`. Comme nous n'avons pas encore configuré le noyau, nous avons besoin de créer ce fichier nous−même pour éviter des échecs de compilation. Créez un fichier `autoconf.h` vide :

```
touch /usr/include/linux/autoconf.h
```

Pourquoi nous copions les en−têtes du noyau et pourquoi nous ne créons pas de liens

Auparavant, une pratique commune consistait à créer des liens symboliques pour les répertoires `/usr/include/{linux,asm}` vers respectivement `/usr/src/linux/include/{linux,asm}`. Ceci est une *mauvaise* idée d'après cet extrait d'un message de Linus Torvalds sur la liste de diffusion du noyau Linux :

```
Je suggère que les personnes qui compilent des noyaux devraient :

 − ne pas créer un seul lien symbolique (sauf celui créé lors de la
construction du noyau, "linux/include/asm" qui est utilisé pour la compilation
du noyau lui−même).

Et oui, c'est ce que je fais. Mon répertoire /usr/src/linux a toujours les
anciens entêtes du noyau 2.2.13, même si je n'ai pas lancé cette version du
noyau depuis un _loong_ moment. Mais Glibc a été compilé avec, donc ces
entêtes correspondent aux objets de la bibliothèque.

Et cela correspond à l'environnement suggéré depuis au moins les cinq dernières
années. Je ne sais pas pourquoi l'idée du lien symbolique est toujours vivante,
comme un mauvais zombie. Pratiquement toutes les distributions conservent
l'idée du lien et tout le monde se souvient que les sources du noyau doivent
aller sous "/usr/src/linux" même si ce n'est plus vrai depuis _trèès_
longtemps.
```

La partie essentielle se trouve là où Linus indique que les fichiers d'entête doivent être *ceux avec lesquels gblic a été compilé*. Ces entêtes doivent être utilisés plus tard lorsque vous compilerez d'autres packages, car ce sont eux qui représentent les fichiers de bibliothèques. En copiant les entêtes, nous nous assurons qu'ils restent disponibles plus tard lors d'une mise à jour du noyau.

Notez qu'il est parfaitement normal d'avoir les sources du noyau dans `/usr/src/linux`, aussi longtemps que vous n'avez pas les liens symboliques `/usr/include/{linux,asm}`.

Installer Man−pages−1.60

```
Estimation du temps de constructionckage:           0,1 SBU
Estimation de l'espace disque requisckage:  15 Mo
```

Contenu de Man−pages

Le package Man−pages contient plus de 1200 pages man. Cette documentation détaille les fonctions C et C++, décrit quelques fichiers périphériques importants et apporte les documents qui seraient autrement être manquants des autres packages.

Fichiers installés : différentes pages man

Dépendances d'installation de Man−pages

Man dépend de Bash, Coreutils, Make.

Installation de Man−pages

Installez Man−pages en lançant :

```
make install
```

Installer Glibc−2.3.2

```
Estimation du temps de construction :           12,3 SBU
Estimation de l'espace disque requis :  784 Mo
```

Contenu de Glibc

(Dernière vérification effectuée auprès de la version 2.3.2.)

Glibc est une bibliothèque C qui apporte les appels système et les fonctions de base telles que open, malloc, printf, etc. La bibliothèque C est utilisée par tous les programmes liés dynamiquement.

Programmes installés : catchsegv, gencat, getconf, getent, glibcbug, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nscd, nscd_nischeck, pcprofiledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump et zic

Bibliothèques installées : ld.so, libBrokenLocale.[a,so], libSegFault.so, libanl.[a,so], libbsd−compat.a, libc.[a,so], libc_nonshared.a, libcrypt.[a,so], libdl.[a,so], libg.a, libieee.a, libm.[a,so], libmcheck.a, libmemusage.so, libnsl.a, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.[a,so], libresolv.[a,so], librpcsvc.a, librt.[a,so], libthread_db.so et libutil.[a,so]

Dépendances d'installation de Glibc

Glibc dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed, Texinfo.

Installation de Glibc

Le système de construction de Glibc est très solide et réalisera l'installation parfaitement même si notre fichier compilateur specs et l'éditeur de liens pointent toujours vers `/tools`. Nous ne pouvons pas ajuster le specs et l'éditeur de liens avant l'installation de Glibc parce que les tests autoconf de Glibc donneraient alors des résultats erronés et feraient ainsi échouer une construction propre.

> **Note :** La suite de tests pour Glibc dans cette section est considérée *critique*. Notre conseil est de la réaliser à tout prix.

Avant de commencer la construction de Glibc, rappelez−vous de déballer Glibc−linuxthreads de nouveau dans le répertoire `glibc−2.3.2` et de supprimer les initialisations de toute variable d'environnement qui surchargerait une option d'optimisation par défaut.

Bien qu'il s'agisse d'un message sans gravité, la phase d'installation de Glibc se plaindra de l'absence de `/etc/ld.so.conf`. Corrigez cet ennuyeux petit message avec :

```
touch /etc/ld.so.conf
```

Ensuite, appliquez le même correctif que celui utilisé précédemment :

```
patch -Np1 -i ../glibc-2.3.2-sscanf-1.patch
```

La documentation de Glibc recommande de construire Glibc en dehors du répertoire des sources, c'est-à-dire dans un répertoire dédié à la construction :

```
mkdir ../glibc-build
cd ../glibc-build
```

Maintenant, préparez la compilation de Glibc :

```
../glibc-2.3.2/configure --prefix=/usr \
    --disable-profile --enable-add-ons \
    --libexecdir=/usr/bin --with-headers=/usr/include
```

La signification des nouvelles options de configure est :

- **`--libexecdir=/usr/bin`** : Ceci fait que le programme `pt_chown` sera installé dans le répertoire `/usr/bin`.
- **`--with-headers=/usr/include`** : Ceci nous assure que les en-têtes du noyau compris dans `/usr/include` sont utilisées pour cette construction. Si vous ne passez pas cette option, alors les en-têtes de `/tools/include` sont utilisées, ce qui n'est évidemment pas idéal (bien qu'ils devraient être identiques). Utiliser cette option présente l'avantage de vous serez informer immédiatement si vous avez installé les en-têtes dans `/usr/include`.

Compilez le paquet :

```
make
```

Testez le résultat :

```
make check
```

Les notes de la suite de tests disponibles dans la section intitulée *Installer Glibc-2.3.2* dans Chapitre 5 sont toujours appropriées ici. Assurez-vous de vous y référer en cas de doute.

Enfin, installez le paquet :

```
make install
```

Les locales qui peuvent permettre à votre système de parler une autre langue, n'ont pas été installées avec la commande ci-dessus. Faites-le ainsi :

```
make localedata/install-locales
```

Une alternative au lancement de la commande précédente est d'installer uniquement les locales dont vous avez besoin ou que vous voulez. Ceci est faisable avec la commande **`localedef`**. Une information là-dessus est disponible dans le fichier `INSTALL` du répertoire `glibc-2.3.2`. Néanmoins, il existe un certain nombre de

locales essentielles pour passer les tests des futurs paquets correctement. Les instructions suivantes, remplaçant la commandes install−locales ci−dessus, installeront l'ensemble minimum de locales nécessaires pour exécuter les tests correctement :

```
mkdir -p /usr/lib/locale
localedef -i de_DE -f ISO-8859-1 de_DE
localedef -i de_DE@euro -f ISO-8859-15 de_DE@euro
localedef -i en_HK -f ISO-8859-1 en_HK
localedef -i en_PH -f ISO-8859-1 en_PH
localedef -i en_US -f ISO-8859-1 en_US
localedef -i es_MX -f ISO-8859-1 es_MX
localedef -i fr_FR -f ISO-8859-1 fr_FR
localedef -i fr_FR@euro -f ISO-8859-15 fr_FR@euro
localedef -i it_IT -f ISO-8859-1 it_IT
localedef -i ja_JP -f EUC-JP ja_JP
```

Puis, construisez les pages man de linuxthreads :

```
make -C ../glibc-2.3.2/linuxthreads/man
```

Et installez−les :

```
make -C ../glibc-2.3.2/linuxthreads/man install
```

Configurer Glibc

Nous avons besoin de créer le fichier `/etc/nsswitch.conf`, parce que, bien que Glibc puisse fournir un paramétrage par défaut lorsque ce fichier manque ou est corrompu, la Glibc ne fonctionne pas correctement pour le réseau. Notre zone horaire a également besoin d'être configurée.

Créez un nouveau fichier`/etc/nsswitch.conf` en exécutant ce qui suit:

```
cat > /etc/nsswitch.conf << "EOF"
# Debut de /etc/nsswitch.conf

passwd: files
group: files
shadow: files

publickey: files

hosts: files dns
networks: files

protocols: db files
services: db files
ethers: db files
rpc: db files

netgroup: db files

# Fin de /etc/nsswitch.conf
EOF
```

Pour trouver votre zone horaire, lancez le script suivant :

```
tzselect
```

Quand vous aurez répondu à quelques questions concernant votre situation géographique, le script vous donnera le nom de votre zone horaire, quelque chose comme *EST5EDT* ou *Canada/Eastern*. Ensuite, créez le fichier `/etc/localtime` en lançant :

```
cp --remove-destination /usr/share/zoneinfo/Canada/Eastern /etc/localtime
```

La signification de l'option est la suivante :

- **`--remove-destination`** : Ceci est nécessaire pour forcer la suppression du lien symbolique existant. La raison de la copie plutôt que la création d'un lien symbolique s'explique par la nécessité de fonctionner même si `/usr` se trouve sur une partition séparée. Ceci peut faire la différence, par exemple lorsque vous démarrez en mode simple utilisateur.

Bien sûr, remplacez *Canada/Eastern* par le nom de la zone horaire que le script **`tzselect`** vous a donné.

Configurer le chargeur dynamique

Par défaut, le chargeur dynamique (`/lib/ld-linux.so.2`) cherche dans `/lib` et `/usr/lib` les bibliothèques dynamiques qui sont nécessaires aux programmes que vous lancez. cependant, s'il y a des bibliothèques dans des répertoires autres que `/lib` et `/usr/lib`, vous devez les ajouter dans le fichier `/etc/ld.so.conf` afin que le chargeur dynamique les trouve. Les deux répertoires couramment utilisés pour recevoir de nouvelles bibliothèques sont `/usr/local/lib` et `/opt/lib`. Pour cette raison, nous devons ajouter ces répertoires dans la liste de recherche du chargeur dynamique.

Créez un nouveau fichier `/etc/ld.so.conf` en exécutant la commande suivante :

```
cat > /etc/ld.so.conf << "EOF"
# Debut de /etc/ld.so.conf

/usr/local/lib
/opt/lib

# Fin /etc/ld.so.conf
EOF
```

Réajuster l'ensemble des outils

Maintenant que les nouvelles bibliothèques C ont été installées, il est temps de réajuster cet ensemble. Nous allons le configurer de façon à ce qu'il lie n'importe quel nouveau programme compilé avec les nouvelles bibliothèques C. De façon simple, ceci est le contraire de ce que nous avions fait à l'étape de "verrouillage" au début de ce chapitre.

La première chose à faire est d'ajuster l'éditeur de liens. Pour ceci, nous retenons les répertoires des sources et de construction de la deuxième passe pour Binutils. Installez l'éditeur de liens ajusté à partir du répertoire `binutils-build` :

```
make -C ld INSTALL=/tools/bin/install install
```

> **Note :** Si vous avez oublié le message d'avertissement précédent pour conserver les répertoires des sources et de construction de Binutils à partir de la seconde passe du chapitre 5 ou si vous les avez supprimé accidentellement ou si vous avez perdu accès à ceux−ci, ne vous inquiétez pas, tout n'est pas perdu. Simplement, ignorez la commande ci−dessus. Cela aura pour résultat que le prochain paquet, Binutils, sera lié avec les bibliothèques Glibc de `/tools` au lieu de `/usr`. Ce n'est pas idéal. Néanmoins, nos tests ont montré que les binaires Binutils devaient être identiques.

A partir de maintenant, tout programme compilé sera *uniquement* lié avec les bibliothèques contenues dans `/usr/lib` et `/lib`. **`INSTALL=/tools/bin/install`** est nécessaire parce que le Makefile créé durant la seconde passe contient toujours la référence à `/usr/bin/install`, que nous n'avons pas encore installé. Certaines distributions hôtes contiennent un lien symbolique `ginstall` qui a la préférence dans le Makefile et donc pourrait proposer un problème ici. La commande ci−dessus s'en occupe aussi.

Vous pouvez maintenant supprimer les répertoires des sources et de construction de Binutils.

La prochaine chose à faire est de modifier le fichier specs de GCC de façon à ce qu'il pointe vers le nouvel éditeur de liens. Comme précédemment, nous utilisons une commande sed pour accomplir ceci :

```
SPECFILE=/tools/lib/gcc-lib/*/*/specs &&
sed -e 's@ /tools/lib/ld-linux.so.2@ /lib/ld-linux.so.2@g' \
    $SPECFILE > newspecfile &&
mv -f newspecfile $SPECFILE &&
unset SPECFILE
```

Encore une fois, un copier/coller de ce qui se trouve ci−dessus est recommandé. Et comme précédemment, ce serait une bonne idée de vérifier le fichier specs pour s'assurer que les changements souhaités ont réellement eu lieu.

> **Important :** Si vous travaillez sur une plateforme où le nom de l'éditeur de liens dynamiques est autre que `ld−linux.so.2`, vous *devez* substituer `ld−linux.so.2` par le nom de l'éditeur de liens dynamiques de votre plate−forme. Référez−vous à la section intitulée *Notes techniques sur l'atelier d'outils* dans Chapitre 5 si nécessaire.

Attention

Il est impératif à ce point de s'arrêter et de s'assurer que les fonctionnalités de base fonctionnent comme prévu. Nous allons effectuer une simple vérification :

```
echo 'main(){}' > dummy.c
gcc dummy.c
readelf -l a.out | grep ': /lib'
```

Si tout a fonctionné correctement, la sortie de la dernière commande sera :

```
[Requesting program interpreter: /lib/ld-linux.so.2]
```

Si vous n'obtenez pas une sortie comme celle montrée ci−dessus, ou aucune sortie du tout, alors quelque chose va très mal. Vous devrez enquêter là−dessus et reprendre chaque étape pour trouver où est situé le problème et le corriger. Il ne sert à rien de continuer jusqu'à ce que ce soit corrigé. Il est probable que quelque chose s'est mal passé avec le fichier specs ci−dessus. Notez particulièrement que `/lib` apparaît maintenant

comme le préfixe de notre éditeur de liens dynamiques. Bien sûr, si vous travaillez sur une plate−forme où le nom de l'éditeur de liens est quelque chose d'autre que `ld−linux.so.2`, alors la sortie sera un peu différente.

Une fois satisfait, nettoyez les fichiers de test :

```
rm dummy.c a.out
```

Installer Binutils−2.14

```
Estimation du temps de compilation :           1,4 SBU
Estimation de l'espace disque requis :  167 Mo
```

Contenu de Binutils

Binutils est une collection d'outils de développement de logiciels contenant un éditeur de liens, un assembleur et d'autres outils pour travailler avec des fichiers objet et des archives.

Programmes installés : addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings et strip

Bibliothèques installés : libiberty.a, libbfd.[a,so] et libopcodes.[a,so]

Dépendances d'installation de Binutils

Binutils dépend de Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Installation de Binutils

C'est le bon moment pour vérifier que vos pseudo−terminaux (PTY) fonctionnent correctement dans l'environnement chroot. Nous allons encore rapidement vérifier que tout est bien configuré en réalisant un simple test :

```
expect −c "spawn ls"
```

Si vous recevez le message :

```
The system has no more ptys.  Ask your system administrator to create more.
```

Votre environnement chroot n'est pas correctement configuré pour les PTY. Dans ce cas, il n'y a aucune raison pour lancer les suites de tests pour Binutils et GCC jusqu'à la résolution de ce problème. Merci de vous référer à la section intitulée *Monter le système de fichier proc et devfs* et la section intitulée *Créer les périphériques (Makedev−1.7)*, et de réaliser les étapes recommandées pour corriger le problème.

> **Note :** La suite de tests pour Binutils dans cette section est considérée comme étant *critique*. Notre conseil est de faire à tout prix ces tests.

Ce paquet est connu pour mal se comporter si vous changez les options d'optimisation par défaut (en incluant les options −march et −mcpu). Donc, si vous avez défini des variables d'environnement qui surchargent les optimisations pas défaut, telles que CFLAGS et CXXFLAGS, nous vous recommandons de supprimer cette initialisation lors de la construction de Binutils.

La documentation de Binutils recommande de construire Binutils en dehors du répertoire des sources, c'est−à−dire dans un répertoire de construction dédié :

```
mkdir ../binutils-build
cd ../binutils-build
```

Maintenant, préparez la compilation de Binutils :

```
../binutils-2.14/configure \
    --prefix=/usr --enable-shared
```

Compilez le paquet :

```
make tooldir=/usr
```

Normalement, *tooldir* (le répertoire où se trouveront les exécutables) est initialisé à $(exec_prefix)/$(target_alias), ce qui sera compris comme, par exemple, `/usr/i686-pc-linux-gnu`. Comme nous ne construisons que notre propre système, nous n'avons pas besoin d'un répertoire spécifique à une cible dans `/usr`. Cette configuration serait utilisée si le système était utilisé pour faire de la cross−compilation (par exemple pour compiler un paquet sur une machine Intel en générant du code exécutable sur des machines à base de PowerPC).

Testez le resultat :

```
make check
```

Les notes de la suite de tests données lors du la section intitulée *Installer Binutils−2.14 − Pass 2* dans Chapitre 5 sont toujours appropriées ici. Assurez−vous de vous y référer si vous avez un quelconque doute.

Installez le paquet :

```
make tooldir=/usr install
```

Installez le fichier d'en−tête *libiberty* nécessaire à certains paquets :

```
cp ../binutils-2.14/include/libiberty.h /usr/include
```

Installer GCC−3.3.1

```
Estimation du temps de construction :           11,7 SBU
Estimation de l'espace disque requis :  294 Mo
```

Contenu de GCC

Le package GCC contient le compilateur GNU, incluant les compilateurs C et C++.

Programmes installés : c++, cc (lien vers gcc), cc1, cc1plus, collect2, cpp, g++, gcc, gccbug et gcov

Bibliothèques installées : libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.[a,so] et libsupc++.a

Dépendances d'installation de GCC

GCC dépend de Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Installation de GCC

> **Note :** La suite de tests pour GCC de cette section est considérée comme *critique*. Notre conseil est de la réaliser quelles que soient les circonstances.

Ce package est connu pour mal se comporter si vous changez les options d'optimisation par défaut (en incluant les options −march et −mcpu). Donc, si vous avez défini des variables d'environnement qui surchargent les optimisations pas défaut, telles que CFLAGS et CXXFLAGS, nous vous recommandons de supprimer cette initialisation lors de la construction de GCC.

Cette fois, nous allons construire le compilateur C et le compilateur C++, donc vous devez déballer les archives tar GCC−core *et* GCC−g++, qui utiliseront le même répertoire. Vous devrez aussi certainement extraire le paquet GCC−testsuite. Le paquet GCC complet contient encore plus de compilateurs. Les instructions de construction pour ceux−ci se trouvent sur http://www.linuxfromscratch.org/blfs/view/stable/general/gcc.html.

```
patch −Np1 −i ../gcc−3.3.1−no_fixincludes−2.patch
patch −Np1 −i ../gcc−3.3.1−suppress−libiberty.patch
```

Le second correctif supprime ici l'installation de libiberty à partir de GCC, car nous utilisons celle fournie par binutils. Faites attention à *ne pas* appliquer le correctif "GCC specs" du chapitre 5.

La documentation GCC recommande de construire GCC en dehors du répertoire source, c'est−à−dire dans un répertoire de construction dédié :

```
mkdir ../gcc−build
cd ../gcc−build
```

Maintenant, préparez la compilation de GCC :

```
../gcc−3.3.1/configure −−prefix=/usr \
    −−enable−shared −−enable−threads=posix \
    −−enable−__cxa_atexit −−enable−clocale=gnu \
    −−enable−languages=c,c++
```

Compilez le package :

```
make
```

Testez le résultat mais ne vous arrêtez pas aux erreurs (vous vous en rappelerez) :

```
make -k check
```

Les notes de la suite de tests de la section intitulée *Installer GCC−3.3.1 − Pass 2* dans Chapitre 5 sont encore une fois appropriées ici. Référerez−vous à celles−ci si vous avez des doutes.

Et, installez le paquet :

```
make install
```

Quelques paquets s'attendent à ce que le préprocesseur C soit installé dans le répertoire `/lib`. Pour utiliser ces paquets, créez ce lien symbolique :

```
ln -s ../usr/bin/cpp /lib
```

Beaucoup de paquets utilisent le nom **`cc`** pour appeler le compilateur C. Pour satisfaire ces paquets, créez le lien symbolique :

```
ln -s gcc /usr/bin/cc
```

> **Note :** Maintenant, il est fortement recommandé de répéter les vérifications réalisées dans ce chapitre. Référez−vous à la section intitulée *Réajuster l'ensemble des outils* et répétez les vérifications. Si les résultats sont mauvais, il est fortement probable que vous avez mal appliqué le correctif "GCC Specs" lors du chapitre 5.

Installer Coreutils−5.0

```
Estimation du temps de construction :           0,9 SBU
Estimation de l'espace disque requis :  69 Mo
```

Contenu de Coreutils

Le package Coreutils contient un grand ensemble d'utilitaires shell basiques.

Programmes installés : basename, cat, chgrp, chmod, chown, chroot, cksum, comm, cp, csplit, cut, date, dd, df, dir, dircolors, dirname, du, echo, env, expand, expr, factor, false, fmt, fold, groups, head, hostid, hostname, id, install, join, kill, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mv, nice, nl, nohup, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, rm, rmdir, seq, sha1sum, shred, sleep, sort, split, stat, stty, su, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, uptime, users, vdir, wc, who, whoami et yes

Dépendances d'installation de Coreutils

Coreutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed.

Installation de Coreutils

Normalement, la fonctionnalité de **uname** est quelque peu cassée, dans le fait que le paramètre **-p** renvoie toujours "unknown". Le correctif suivant corrige ce comportement pour les architectures Intel :

```
patch -Np1 -i ../coreutils-5.0-uname.patch
```

Nous ne voulons pas que Coreutils installe sa version du programme **hostname** parce qu'il est inférieur à la version fournie par Net-tools. Empêchez cette installation en appliquant un correctif :

```
patch -Np1 -i ../coreutils-5.0-hostname-2.patch
```

Maintenant, préparez la compilation de Coreutils :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Le programme **su** de Coreutils n'était pas installé parce qu'il avait besoin des privilèges de *root* pour le faire. Nous en avons besoin quelques instants pour la suite de tests. Donc, nous contournons le problème en l'installant maintenant :

```
make install-root
```

Ce paquet dispose d'une suite de tests réalisant un ensemble de vérifications pour s'assurer que tout est construit correctement. Néanmoins, cette suite de tests particuliers fait quelques suppositions sur la présence d'utilisateurs et de groupes non root qui ne s'appliquent pas aussi tôt dans la construction de LFS. Nous créons donc un utilisateur système dummy et deux groupes dummy pour permettre aux tests de s'exécuter correctement. Si vous décidez de ne pas lancer cette suite de tests, passez directement à la section "Installez le paquet". Les commandes suivantes prépareront la suite de tests. Créez l'utilisateur et les deux groupes :

```
echo "dummy1:x:1000" >> /etc/group
echo "dummy2:x:1001:dummy" >> /etc/group
echo "dummy:x:1000:1000:::/bin/bash" >> /etc/passwd
```

Certains tests doivent être lancés en tant que *root* :

```
make check-root
```

Le reste des tests est lancé en tant qu'utilisateur *dummy* :

```
su dummy -c "make RUN_EXPENSIVE_TESTS=yes check"
```

Supprimez les noms d'utilisateur et de groupe dummy :

```
sed -i.bak '/dummy/d' /etc/passwd /etc/group
```

Installez le paquet :

```
make install
```

Et déplacez quelques programmes vers leur bon emplacement :

```
mv /usr/bin/{basename,cat,chgrp,chmod,chown,cp,dd,df} /bin
mv /usr/bin/{dir,dircolors,du,date,echo,false,head} /bin
mv /usr/bin/{install,ln,ls,mkdir,mkfifo,mknod,mv,pwd} /bin
mv /usr/bin/{rm,rmdir,shred,sync,sleep,stty,su,test} /bin
mv /usr/bin/{touch,true,uname,vdir} /bin
mv /usr/bin/chroot /usr/sbin
```

Finalement, créez quelques liens symboliques nécessaires :

```
ln -s test /bin/[
ln -s ../../bin/install /usr/bin
```

Installer Zlib−1.1.4

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  1,5 Mo
```

Contenu de Zlib

Le package Zlib contient la bibliothèque zlib, utilisé par certains programmes pour ses fonctions de compression et de décompression.

Bibliothèques installées : libz[a,so]

Dépendances d'installation de Zlib

Zlib dépend de Binutils, Coreutils, GCC, Glibc, Make, Sed.

Installation de Zlib

Zlib a un problème de dépassement de tampon potentiel dans sa fonction *gzprintf()*, qui, bien que difficile à être exploitée, devrait être corrigée avec l'application de ce correctif :

```
patch -Np1 -i ../zlib-1.1.4-vsnprintf.patch
```

Maintenant, préparez la compilation de Zlib :

```
./configure --prefix=/usr --shared
```

Note: Zlib est connu pour construire sa bibliothèque partagée de façon incorrecte si un CFLAGS est spécifié dans son environnement. Si vous utilisez vos propres variables CFLAGS, assurez−vous d'ajouter la directive ***−fPIC*** à cette étape et de la supprimer après.

Compilez le paquet :

```
make
```

Installez les bibliothèques partagées :

```
make install
```

Maintenant, construisez aussi les bibliothèques non partagées :

```
make clean
./configure --prefix=/usr
make
```

Ce pacquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make test
```

Enfin, installez le paquet :

```
make install
```

La bibliothèque partagée Zlib devrait être installée dans le répertoire `/lib`. De cette façon, au cas où vous devriez redémarrer sans le répertoire `/usr`, les programmes systèmes vitaux auront toujours accès à la bibliothèque :

```
mv /usr/lib/libz.so.* /lib
```

Le lien symbolique `/usr/lib/libz.so` pointe vers un fichier qui n'existe plus parce que nous l'avons déplacé. Créez un lien symbolique vers le nouvel emplacement de la bibliothèque :

```
ln -sf ../../lib/libz.so.1 /usr/lib/libz.so
```

Zlib n'installe pas sa page man. Lancez cette commande pour installer cette documentation :

```
cp zlib.3 /usr/share/man/man3
```

Installer Lfs−Utils−0.3

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  1,1 Mo
```

Contenu de Lfs−Utils

Le package Lfs−Utils contient quelques programmes divers provenant de plusieurs packages mais qui ne sont pas assez importants pour disposer de leur propre package.

Programmes installés : mktemp, tempfile, http−get et iana−net

Fichiers installés : protocols, services

Dépendances d'installation de Lfs−Utils

(Les dépendances n'ont pas encore été vérifié.)

Installation de Lfs−Utils

Compilez le paquet :

```
make
```

Et installez−le :

```
make install
```

Maintenant, copiez les deux fichiers inclus dans l'archive tar Lfs−Utils vers leur destination :

```
cp etc/{services,protocols} /etc
```

Le fichier `/etc/services` est utilisé pour convertir les numéros de services en noms lisibles par des humains alors que le fichier `/etc/protocols` fait la même chose pour les numéros de protocole.

Installer Findutils−4.1.20

```
Estimation du temps de compilation :           0,2 SBU
Estimation de l'espace disque requis :  7,5 Mo
```

Contenu de Findutils

Le package Findutils contient les programmes de recherche de fichiers, soit en direct (en faisant une recherche récursive en direct à travers les répertoires et seulement en affichant des fichiers qui remplissent ses spécifications) soit en cherchant à travers une base de données.

Programmes installés : bigram, code, find, frcode, locate, updatedb et xargs

Dépendances d'installation de Findutils

Findutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Installer Findutils

Préparez la compilation de Findutils :

```
./configure --prefix=/usr --libexecdir=/usr/bin
```

Par défaut, la base de données updatedb se trouve dans `/usr/var`. Pour rendre l'emplacement du fichier `/var/lib/misc/locatedb` compatible avec FHS, utilisez l'option *−−localstatedir=/var/lib/misc* de **`configure`**.

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Et installez le paquet :

```
make install
```

Installer Gawk−3.1.3

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  17 Mo
```

Contenu de Gawk

Gawk est une implémentation de awk qui est utilisée pour manipuler des fichiers texte.

Programmes installés : awk (lien vers gawk), gawk, gawk−3.1.3, grcat, igawk, pgawk, pgawk−3.1.3 et pwcat.

Dépendances d'installation de Gawk

Gawk dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Installation de Gawk

Tout d'abord, appliquez un correctif pour résoudre les problèmes suivants :

- L'emplacement par défaut de Gawk pour certains de ses exécutables est `$prefix/libexec/awk`. Cet emplacement n'est pas compatible avec FHS, qui ne mentionne à aucun moment un répertoire nommé `libexec`. Le correctif rend possible le passage d'une option *−−libexecdir* au script configure, de façon à ce que nous utilisions un emplacement plus approprié pour les binaires **`grcat`** et **`pwcat`** : `/usr/bin`.
- Le répertoire des données par défaut de Gawk est `$prefix/share/awk`. Mais les répertoires spécifiques aux paquets devraient être nommés suivant le nom du paquet et son numéro de version (par exemple, `gawk−7.7.2.`) et non pas simplement par son nom de paquet car il pourrait exister plusieurs versions d'un paquet installé sur le système. Le correctif modifie le nom du répertoire de données en un `$prefix/share/gawk−3.1.3` plus correct.
- Le correctif nous assure aussi que le répertoire des données et son contenu sont supprimés avec un *make uninstall*.

```
patch −Np1 −i ../gawk−3.1.3−libexecdir.patch
```

Maintenant, préparez la compilation de Gawk :

```
./configure --prefix=/usr --libexecdir=/usr/bin
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests qui s'assure que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Enfin, installez le paquet :

```
make install
```

Installer Ncurses−5.3

```
Estimation du temps de construction :           0,6 SBU
Estimation de l'espace disque requis :  27 Mo
```

Contenu de Ncurses

(Dernière version effectuée auprès de la version 5.3.)

Le package Ncurses apporte des bibliothèques de gestion de caractères et de terminaux, incluant les panneaux et les menus.

Programmes installés : captoinfo (lien vers tic), clear, infocmp, infotocap (lien vers tic), reset (lien vers tset), tack, tic, toe, tput et tset

Bibliothèques installées : libcurses.[a,so] (lien vers libncurses.[a,so]), libform.[a,so], libmenu.[a,so], libncurses++.a, libncurses.[a,so], libpanel.[a,so]

Dépendances d'installation de Ncurses

Ncurses dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Installation de Ncurses

Tout d'abord, corrigez deux petits bogues :

```
patch -Np1 -i ../ncurses-5.3-etip-2.patch
patch -Np1 -i ../ncurses-5.3-vsscanf.patch
```

Le premier correctif s'occupe du fichier d'en−tête `etip.h` alors que le second empêche les messages d'avertissement dû à l'utilisation d'en−têtes obsolètes.

Maintenant, préparez la compilation de Ncurses :

```
./configure --prefix=/usr --with-shared \
    --without-debug
```

Compilez le paquet :

```
make
```

Installez le paquet :

```
make install
```

Donnez le droit d'exécution pour les bibliothèques Ncurses :

```
chmod 755 /usr/lib/*.5.3
```

Et corrigez une bibliothèque qui ne devrait pas être exécutable :

```
chmod 644 /usr/lib/libncurses++.a
```

Déplacez les bibliothèques dans le répertoire `/lib`, où elles devraient se trouver :

```
mv /usr/lib/libncurses.so.5* /lib
```

Comme les bibliothèques ont été déplacées dans `/lib`, quelques liens symboliques pointent actuellement vers des fichiers inexistants. Recréez ces liens :

```
ln -sf ../../lib/libncurses.so.5 /usr/lib/libncurses.so
ln -sf libncurses.so /usr/lib/libcurses.so
```

Installer Vim-6.2

```
Estimation du temps de construction :           0,4 SBU
Estimation de l'espace disque requis :  34 Mo
```

Alternatives à Vim

Si vous préférez utiliser un autre éditeur, comme Emacs, Joe ou Nano, au lieu de Vim, jetez un oeil sur http://www.linuxfromscratch.org/blfs/view/stable/postlfs/editors.html pour des instructions d'installation.

Contenu de Vim

Le package Vim contient un éditeur de texte configurable construit pour permettre une édition efficace du texte.

Programmes installés : efm_filter.pl, efm_perl.pl, ex (lien vers vim), less.sh, mve.awk, pltags.pl, ref, rview (lien vers vim), rvim (lien vers vim), shtags.pl, tcltags, vi (lien vers vim), view (lien vers vim), vim, vim132, vim2html.pl, vimdiff (lien vers vim), vimm, vimspell.sh, vimtutor et xxd

Dépendances d'installation de Vim

Vim dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Installation de Vim

Modifiez les emplacements par défaut des fichiers `vimrc` et `gvimrc` pour `/etc`.

```
echo '#define SYS_VIMRC_FILE "/etc/vimrc"' >> src/feature.h
echo '#define SYS_GVIMRC_FILE "/etc/gvimrc"' >> src/feature.h
```

Maintenant, préparez la compilation de Vim :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Et installez le paquet :

```
make install
```

Vim peut s'exécuter dans l'ancien mode *vi* en créant un lien symbolique, avec la commande suivante :

```
ln -s vim /usr/bin/vi
```

Si vous pensez installer le système X Window sur votre système LFS, vous pourriez vouloir recompiler Vim après avoir installé X. Vim dispose d'une version GUI très agréable de l'éditeur, mais nécessitant l'installation de X et de quelques autres bibliothèques. Pour plus d'informations, lisez la documentation de Vim.

Configurer Vim

Par défault, vim fonctionne en mode compatible vi. Certaines personnes l'aiment comme cela, mais nous avons une forte préférence pour faire tourner Vim en mode Vim (autrement nous n'aurions pas inclus vim dans ce livre mais l'original vi). Créez `/root/.vimrc` en exécutant ceci :

```
cat > /root/.vimrc << "EOF"
" Debut de /root/.vimrc

set nocompatible
set bs=2

" Fin de /root/.vimrc
EOF
```

Installer M4−1.4

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  3,0 Mo
```

Contenu de M4

M4 est un processeur de macro. Il copie l'entrée vers la sortie, exécutant les macros qu'il trouve. Les macros sont soit intégrées soit définies par l'utilisateur, et peuvent prendre n'importe quel nombre d'arguments. En plus d'exécuter certaines macros, m4 dispose de fonctions intégrées permettant d'inclure des fichiers nommés, de lancer des commandes Unix, de réaliser quelques calculs arithmétiques, de manipuler du texte de différentes façons, d'utiliser la récursivité, etc... Le programme m4 est utilisé soit comme interface pour un compilateur soit directement comme un processeur de macros.

Programme installé : m4

Dépendances d'installation de M4

M4 dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed.

Installation de M4

Préparez la compilation de M4 :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Enfin, installez le paquet :

```
make install
```

Installer Bison−1.875

```
Estimation du temps de compilation :           0,6 SBU
Estimation de l'espace disque requis :  10,6 Mo
```

Contenu de Bison

Bison est un générateur de l'analyseur, un remplacement pour yacc. Bison génère un programme analysant la structure d'un fichier texte.

Programmes installés : bison et yacc

Bibliothèques installés : liby.a

Dépendances d'installation de Bison

Bison dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make, Sed.

Installation de Bison

Tout d'abord, nous utilisons un correctif pour bison, récupéré du CVS, car il corrige un problème de compilation mineur avec certains paquets :

```
patch −Np1 −i ../bison−1.875−attribute.patch
```

Préparez la compilation de Bison :

```
./configure −−prefix=/usr
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Et installez le paquet :

```
make install
```

Installer Less−381

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  3,4 Mo
```

Contenu de Less

Le programme less est un paginateur de fichier (ou un afficheur de texte). Il affiche le contenu d'un fichier et permet le défilement du texte. Less dispose de quelques fonctionnalités qui ne sont pas incluses dans le paginateur << more >>, telle que le défilement arrière.

Programmes installés : less, lessecho et lesskey

Dépendances d'installation de Less

Less dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Installation de Less

Préparez la compilation de Less :

```
./configure --prefix=/usr --bindir=/bin --sysconfdir=/etc
```

La signification de l'option configure est :

- **`--sysconfdir=/etc`** : Cette option indique aux programmes créés par le package de rechercher leur fichiers de configuration dans `/etc`.

Compilez le paquet :

```
make
```

Et installez−le :

```
make install
```

Installer Groff−1.19

```
Estimation du temps de construction :           0,5 SBU
Estimation de l'espace disque requis :  43 Mo
```

Contenu de Groff

Le package Groff inclut plusieurs programmes pour traiter du texte et le formatter. Groff traduit du texte et des commandes spécifiques en une sortie formattée, telle que ce que vous voyez dans une page man.

Programmes installés : addftinfo, afmtodit, eqn, eqn2graph, geqn (lien vers eqn), grn, grodvi, groff, groffer, grog, grolbp, grolj4, grops, grotty, gtbl (lien vers tbl), hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pfbtops, pic, pic2graph, post−grohtml, pre−grohtml, refer, soelim, tbl, tfmtodit, troff et zsoelim (lien vers soelim)

Dépendances d'installation de Groff

Groff dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Installation de Groff

Groff s'attend à ce que la variable d'environnement PAGE contienne la taille par défaut du papier. Pour ceux habitant aux Etats−Unis, la commande ci−dessous est appropriée. Si vous vivez ailleurs, vous pouvez changer *PAGE=letter* par *PAGE=A4*.

Préparez la compilation de Groff :

```
PAGE=letter ./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Puis, installez-le :

```
make install
```

Quelques programmes de documentation, tels que **xman**, ne fonctionneront pas correctement sans les liens symboliques suivants :

```
ln -s soelim /usr/bin/zsoelim
ln -s eqn /usr/bin/geqn
ln -s tbl /usr/bin/gtbl
```

Installer Sed-4.0.7

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  5,2 Mo
```

Contenu de Sed

sed est un éditeur en flux. Un éditeur en flux est utilisé pour réaliser des transformations de texte basique sur une entrée en flux (un fichier ou une entrée à partir d'un tuyau).

Programme installé : sed

Dépendances d'installation de Sed

(Dernière vérification effectuée auprès de la version 3.02.)

Sed dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Texinfo.

Installation de Sed

Préparez la compilation de Sed :

```
./configure --prefix=/usr --bindir=/bin
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Enfin, installez le paquet :

```
make install
```

Installer Flex−2.5.4a

```
Estimation du temps de compilation :           0,1 SBU
Estimation de l'espace disque requis :  3,4 Mo
```

Contenu de Flex

Le package Flex est utilisé pour générer des programmes de reconnaissance de modèles dans du texte.

Programmes installés : flex, flex++ (lien vers flex) et lex

Bibliothèque installée : libfl.a

Dépendances d'installation de Flex

Flex dépend de Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make, Sed.

Installation de Flex

Préparez la compilation de Flex :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make bigcheck
```

Et installez le paquet :

```
make install
```

Certains paquets s'attendent à trouver la bibliothèque Lex dans `/usr/lib`. Créez un lien symbolique pour prendre ceci en compte :

```
ln -s libfl.a /usr/lib/libl.a
```

Quelques programmes ne connaissent pas encore **`flex`** et essaient de lancer son prédécesseur **`lex`**. Pour

supporter ces programmes, créer un script shell nommé `lex` appelant **`flex`** dans son mode d'émulation Lex :

```
cat > /usr/bin/lex << "EOF"
#!/bin/sh
# Debut de /usr/bin/lex

exec /usr/bin/flex -l "$@"

# Fin /usr/bin/lex
EOF
chmod 755 /usr/bin/lex
```

Installer Gettext-0.12.1

```
Estimation du temps de construction :           6,9 SBU
Estimation de l'espace disque requis :  55 Mo
```

Contenu de Gettext

Le package Gettext est utilisé pour l'internationalisation et la localisation. Les programmes peuvent êtres compilés avec le support de la langue native ('Native Language Support' ou NLS) qui leur permettent d'afficher les messages dans la langue native de l'utilisateur.

Programmes installés : autopoint, config.charset, config.rpath, gettext, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, project-id, team-address, trigger, urlget, user-email et xgettext

Bibliothèques installées : libasprintf[a,so], libgettextlib[a,so], libgettextpo[a,so] et libgettextsrc[a,so]

Dépendances d'installation de Gettext

Gettext dépend de Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Installation de Gettext

Préparez la compilation de Gettext :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Enfin, installez le package :

```
make install
```

Installer Net−tools−1.60

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  9,4 Mo
```

Contenu de Net−tools

Le package Net−tools contient une collection de programmes formant la base réseau sous Linux.

Programmes installés : arp, dnsdomainname (lien vers hostname), domainname (lien vers hostname), hostname, ifconfig, nameif, netstat, nisdomainname (lien vers hostname), plipconfig, rarp, route, slattach et ypdomainname (lien vers hostname)

Dépendances d'installation de Net−tools

Net−tools dépend de Bash, Binutils, Coreutils, GCC, Glibc, Make.

Installation de Net−tools

Si vous ne savez quoi répondre à toutes les questions posées pendant la phase du **`make config`** ci−dessous, acceptez les valeurs par défaut. Elles sont tout à fait bien dans la majorité des cas. Un ensemble de questions vous est posé sur les protocoles réseau que vous avez activé sur votre noyau. Les réponses par défaut feront que tous les outils de ce package fonctionneront avec les protocoles les plus fréquents : TCP, PPP et quelques autres encore. Vous aurez toujours besoin d'activer réellement ces protocoles dans le noyau −− vous devez donc dire simplement au package d'inclure ces protocoles dans ces programmes, mais c'est au noyau de rendre les protocoles disponibles.

Tout d'abord, corrigez un léger problème de syntaxe dans les sources avec le programme mii−tool :

```
patch −Np1 −i ../net−tools−1.60−miitool−gcc33−1.patch
```

Maintenant, préparez la compilation de Net−tools avec :

```
make config
```

Si vous souhaitez accepter les paramètrages par défaut, vous pouvez passer les questions générées par *make config* en lançant **`yes "" | make config`** à la place.

Compilez le paquet :

```
make
```

Et installez−le :

```
make update
```

Installer Inetutils−1.4.2

```
Estimation du temps de construction :  0,2 SBU
Estimation de l'espace disque requis :  11 Mo
```

Contenu de Inetutils

Le package Inetutils contient des clients et des serveurs réseau.

Programmes installés : ftp, ping, rcp, rlogin, rsh, talk, telnet et tftp

Dépendance d'installation d'Inetutils

Inetutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Installation de Inetutils

Préparez la compilation d'Inetutils :

```
./configure --prefix=/usr --disable-syslogd \
    --libexecdir=/usr/sbin --disable-logger \
    --sysconfdir=/etc --localstatedir=/var \
    --disable-whois --disable-servers
```

La signification des options de configure est :

- **`--disable-syslogd`** : Cette option empêche inetutils d'installer le démon de journalisation du système, installé avec le package Sysklogd.
- **`--disable-logger`** : Cette option empêche inetutils d'installer le programme logger, utilisé par des scripts pour passer des messages au démon de journalisation du système. Nous ne l'installons pas car Util−linux installe une version bien supérieure.
- **`--disable-whois`** : Cette option désactive la construction du client whois d'inetutils, qui est affreusement obsolète. Des instructions pour un meilleur client whois se trouvent dans le livre BLFS.
- **`--disable-servers`** : Ceci désactive l'installation de différents serveurs réseau inclus dans le package Inetutils. Ces serveurs ne sont absolument pas appropriés dans un système LFS de base. Certains sont non sécurisés par nature et sont seulement considérés sûr sur des réseaux de confiance. Plus d'information est disponible sur http://www.linuxfromscratch.org/blfs/view/stable/basicnet/inetutils.html. Notez que des programmes de substitution bien supérieurs sont disponibles pour un bon nombre de ces serveurs.

Compilez le paquet :

```
make
```

Installez−le :

```
make install
```

Et déplacez le programme **`ping`** au bon emplacement :

```
mv /usr/bin/ping /bin
```

Installer Perl−5.8.0

```
Estimation du temps de construction :           2,9 SBU
Estimation de l'espace disque requis :  143 Mo
```

Contenu de Perl

Le package Perl contient perl, l'acronyme de Practical Extraction and Report Language (Langage pratique d'extraction et de rapport). Perl combine certaines des meilleurs fonctionnalités des langages C, sed, awk et sh en un seul langage extrêmement puissant.

Programmes installés : a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (lien vers perl), perlbug, perlcc, perldoc, perlivp, piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, psed (lien vers s2p), pstruct (lien vers c2ph), s2p, splain and xsubpp

Bibliothèques installées : (trop nombreux pour les nommer toutes)

Dépendances d'installation de Perl

Perl dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Installation de Perl

Préparez la compilation de Perl :

```
./configure.gnu --prefix=/usr
```

Si vous voulez plus de contrôle sur la façon dont Perl se configure pour se contruire, vous pouvez lancer le script interactif **`Configure`** à la place et modifier ainsi la façon dont Perl est construit. Si vous pensez que vous pouvez vivre avec les valeurs par défaut (corrects) que Perl détecte automatiquement, alors utilisez simplement la commande ci−dessous.

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests réalisant un certain nombre de tests pour s'assurer que tout est construit correctement. Si vous décidez de le lancer, vous devez tout d'abord créé un simple fichier `/etc/hosts`, nécessaire pour que quelques tests puissent comprendre le nom *localhost*:

```
echo "127.0.0.1 localhost $(hostname)" > /etc/hosts
```

Maintenant, lancez les tests, si vous le souhaitez :

```
make test
```

Enfin, installez le paquet :

```
make install
```

Installer Texinfo−4.6

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  17 Mo
```

Contenu de Texinfo

Le package Texinfo contient des programmes utilisés pour lire, écrire et convertir des documents Info, qui apporte une documentation système.

Programmes installés : info, infokey, install−info, makeinfo, texi2dvi et texindex

Dépendances d'installation de Texinfo

Texinfo dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed.

Installation de Texinfo

Préparez la compilation de Texinfo :

```
./configure --prefix=/usr
```

Compilez le package :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Installez le paquet :

```
make install
```

Et installez de façon optionnelle les composants appartenant à une installation TeX :

```
make TEXMF=/usr/share/texmf install-tex
```

La signification du paramètre de make est :

- **`TEXMF=/usr/share/texmf`** : La variable TEXMF de make contient l'emplacement de la racine de TeX si, par exemple, vous installez le package TeX plus tard.

Installer Autoconf−2.57

```
Estimation du temps de compilation :           2,9 SBU
Estimation de l'espace disque requis :  7,7 Mo
```

Contenu d'Autoconf

Autoconf crée des scripts shell qui configurent automatiquement du code source.

Programmes installés : autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate et ifnames

Dépendances d'installation d'Autoconf

Autoconf dépend de Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

Installation d'Autoconf

Préparez la compilation d'Autoconf :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Et installez le paquet :

```
make install
```

Installer Automake−1.7.6

```
Estimation du temps de compilation :           5,3 SBU
Estimation de l'espace disque requis :  6,8 Mo
```

Contenu d'Automake

Automake génère des fichiers Makefile.in, à utiliser avec Autoconf.

Programmes installés : acinstall, aclocal, aclocal−1.6, automake, automake−1.6, compile, config.guess, config.sub, depcomp, elisp−comp, install−sh, mdate−sh, missing, mkinstalldirs, py−compile, ylwrap

Dépendances d'installation d'Automake

Automake dépend de Autoconf, Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

Installation de Automake

Préparez la compilation d'Automake :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Installez le paquet :

```
make install
```

Et, créez le lien symbolique nécessaire :

```
ln -s automake-1.7 /usr/share/automake
```

Installer Bash-2.05b

```
Estimation du temps de compilation :           1,2 SBU
Estimation de l'espace disque requis :  27 Mo
```

Contenu de Bash

Bash, acronyme de Bourne-Again SHell, est un interpréteur de commande très répandu sur les systèmes Unix. Il se charge d'interpréter et d'évaluer les commandes que l'utilisateur entre sur l'entrée standard (le clavier), comme par exemple lancer un programme.

Programmes installés : bash, sh (lien vers bash) et bashbug

Dépendances d'installation de Bash

Bash dépend de Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Sed.

Installation de Bash

Bash contient un certain nombres de bogues faisant qu'il ne se comporte pas de la façon attendue. Corrigez ce comportement avec le correctif suivant :

```
patch −Np1 −i ../bash−2.05b−2.patch
```

Préparez la compilation de Bash :

```
./configure −−prefix=/usr −−bindir=/bin
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make tests
```

Installez le paquet :

```
make install
```

Et rechargez le nouveau programme **bash** :

```
exec /bin/bash −−login +h
```

Installer File−4.04

```
Estimation du temps de compilation :           0,1 SBU
Estimation de l'espace disque requis :  6,3 Mo
```

Contenu de File

File est un utilitaire servant à déterminer les types de fichiers.

Programme installé : file

Bibliothèque installée : libmagic.[a,so]

Dépendances d'installation de File

File dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, Zlib.

Installation de File

Préparez la compilation de File :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Et, installez−le :

```
make install
```

Installer Libtool−1.5

```
Estimation du temps de construction :           1,5 SBU
Estimation de l'espace disque requis :  20 Mo
```

Contenu de Libtool

GNU libtool est un outil générique de support de scripts. Libtool cache la complexité en utilisant les bibliothèques partagées derrière une interface portable cohérente.

Programmes installés : libtool et libtoolize

Bibliothèques installées : libltdl.[a,so]

Dépendances d'installation de Libtool

Libtool dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation de Libtool

Préparez la compilation de Libtool :

```
./configure --prefix=/usr
```

Compilez le package :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Enfin, installez le paquet :

```
make install
```

Installer Bzip2−1.0.2

```
Estimation du temps de compilation :           0.1 SBU
Estimation de l'espace disque requis :  3.0 Mo
```

Contenu de Bzip2

Bzip2 est un compresseur de fichiers qui arrive habituellement à une meilleure compression que ne le fait **`gzip`** traditionel.

Programmes installés : bunzip2 (lien vers bzip2), bzcat (lien vers bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless et bzmore

Bibliothèques installées : libbz2.a, libbz2.so (lien vers libbz2.so.1.0), libbz2.so.1.0 (lien vers libbz2.so.1.0.2) et libbz2.so.1.0.2

Dépendances d'installation de Bzip2

Bzip2 dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make.

Installation de Bzip2

Préparez la compilation de Bzip2 avec :

```
make −f Makefile−libbz2_so
make clean
```

L'option *−f* fait que Bzip2 est construit avec un fichier `Makefile` différent, dans ce cas le fichier `Makefile−libbz2_so`, qui crée une bibliothèque partagée `libbz2.so` et la lie aux utilitaires.

Compilez le paquet :

```
make
```

Installez−le :

```
make install
```

Et installez le binaire **`bzip2`** dans le répertoire `/bin` en ajoutant les liens symboliques nécessaires. Puis, faites un peu de ménage :

```
cp bzip2−shared /bin/bzip2
cp −a libbz2.so* /lib
ln −s ../../lib/libbz2.so.1.0 /usr/lib/libbz2.so
rm /usr/bin/{bunzip2,bzcat,bzip2}
mv /usr/bin/{bzip2recover,bzless,bzmore} /bin
ln −s bzip2 /bin/bunzip2
ln −s bzip2 /bin/bzcat
```

Installer Diffutils−2.8.1

```
Estimation du temps de compilation :           0,1 SBU
Estimation de l'espace disque requis :  7,5 Mo
```

Contenu de Diffutils

Les programmes de ce package vous montre les différences entre les deux fichiers ou répertoires. Son utilisation la plus commune est la création de correctifs.

Programmes installés : cmp, diff, diff3 et sdiff

Dépendances d'installation de Diffutils

Diffutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Installation de Diffutils

Préparez la compilation de Diffutils :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Et installez−le :

```
make install
```

Installer Ed−0.2

```
Estimation du temps de compilation :           0,1 SBU
Estimation de l'espace disque requis :  3,1 Mo
```

Contenu d'Ed

GNU ed est un éditeur par lignes compatible POSIX.

Programmes installés : ed et red (lien vers ed)

Dépendances d'installation de Ed

Ed dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation de Ed

> **Note :** Ed n'est pas un programme que beaucoup de gens utilisent. Il est installé parce qu'il peut être utilisé par le programme patch si vous rencontrez un correctif basé sur ed. Ceci arrive rarement parce que les correctifs basés sur diff sont préférés de nos jours.

Ed utilise normalement la fonction mktemp pour créer des fichiers temporaires dans `/tmp`, mais cette fonctionnalité contient une vulnérabilité (voir la section sur les fichiers temporaires dans http://en.tldp.org/HOWTO/Secure−Programs−HOWTO/avoid−race.html). Le correctif suivant fait que Ed utilise mkstemp à la place, ce qui est la façon recommandée de créer des fichiers temporaires.

Appliquez le correctif :

```
patch −Np1 −i ../ed−0.2−mkstemp.patch
```

Maintenant, préparez la compilation de Ed :

```
./configure −−prefix=/usr
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Installez le paquet :

```
make install
```

Et, déplacez les programmes vers le répertoire `/bin` de façon à ce qu'ils puissent être utilisés au cas où la partition `/usr` ne serait pas disponible.

```
mv /usr/bin/{ed,red} /bin
```

Installer Kbd−1.08

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  12 Mo
```

Contenu de Kbd

Kbd contient des fichiers de codage des touches et des utilitaires pour le clavier.

Programmes installés : chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, getunimap, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (lien vers psfxtable), psfgettable (lien vers psfxtable), psfstriptable (lien vers psfxtable), psfxtable, resizecons, setfont, setkeycodes, setleds, setlogcons, setmetamode, setvesablank, showconsolefont, showkey, unicode_start et unicode_stop

Dépendances d'installation de Kbd

Kbd dépend de Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, Gzip, M4, Make, Sed.

Installation de Kbd

Par défaut, certains des utilitaires de Kbd (**`setlogcons`**, **`setvesablank`** et **`getunimap`**) ne sont pas installés. Tout d'abord, activez la compilation de ces utilitaires :

```
patch -Np1 -i ../kbd-1.08-more-programs.patch
```

Maintenant, préparez la compilation de Kbd :

```
./configure
```

Compilez le paquet :

```
make
```

Et installez-le :

```
make install
```

Installer E2fsprogs-1.34

```
Estimation du temps de compilation :           0,6 SBU
Estimation de l'espace disque requis :  48,4 Mo
```

Contenu de E2fsprogs

E2fsprogs apporte des utilitaires du système de fichiers ext2. Il supporte aussi le système de fichiers ext3 avec le support de la journalisation.

Programmes installés : badblocks, blkid, chattr, compile_et, debugfs, dumpe2fs, e2fsck, e2image, e2label, findfs, fsck, fsck.ext2, fsck.ext3, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mklost+found, resize2fs, tune2fs et uuidgen.

Bibliothèques installées : libblkid.[a,so], libcom_err.[a,so], libe2p.[a,so], libext2fs.[a,so], libss.[a,so] et libuuid.[a,so]

Dépendances d'installation d'E2fsprogs

E2fsprogs dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo.

Installation de E2fsprogs

Il est recommandé de construire E2fsprogs en dehors du répertoire des sources :

```
mkdir ../e2fsprogs-build
cd ../e2fsprogs-build
```

Préparez la compilation d'E2fsprogs :

```
../e2fsprogs-1.34/configure --prefix=/usr --with-root-prefix="" \
    --enable-elf-shlibs
```

La signification des options de configure est :

- **`--with-root-prefix=""`** : Certains programmes (tel que le programme e2fsck) sont considérés comme essentiels. Quand, par exemple, `/usr` n'est pas monté, ces programmes essentiels doivent tout de même être accessibles. Ils appartiennent à des répertoires comme `/lib` et `/sbin`. Si cette option n'est pas passé au script configure d'E2fsprogs, les programmes seront placés dans le répertoire, ce qui n'est pas ce que nous souhaitons.
- **`--enable-elf-shlibs`** : Ceci crée les bibliothèques partagées que certains des programmes de ce paquet utilisent.

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Installez la plupart du paquet :

```
make install
```

Et installez aussi les bibliothèques partagées :

```
make install-libs
```

Installer Grep-2.5.1

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  5,8 Mo
```

Contenu de Grep

Grep est un programme utilisé pour imprimer des lignes d'un fichier ressemblant à un modèle spécifié.

Programmes installés : egrep (lien vers grep), fgrep (lien vers grep) et grep

Dépendances d'installation de Grep

Grep dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Sed, Texinfo.

Installation de Grep

Préparez la compilation de Grep :

```
./configure --prefix=/usr --bindir=/bin \
    --with-included-regex
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Enfin, installez le paquet :

```
make install
```

Installer Grub−0.93

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  10 Mo
```

Contenu de Grub

Le package Grub contient un chargeur de démarrage.

Programmes installés : grub, grub−install, grub−md5−crypt, grub−terminfo et mbchk

Dépendances d'installation de Grub

Grub dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Installation de Grub

Ce package est connu pour mal se comporter si vous changez les options d'optimisation par défaut (en incluant les options −march et −mcpu). Donc, si vous avez défini des variables d'environnement qui surchargent les optimisations pas défaut, telles que CFLAGS et CXXFLAGS, nous vous recommendons de supprimer cette initialisation lors de la construction de Grub.

Tout d'abord, corrigez un problème de compilation avec GCC−3.3.1 :

```
patch -Np1 -i ../grub-0.93-gcc33-1.patch
```

Maintenant, préparez la compilation de Grub :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Et installez-le :

```
make install
mkdir /boot/grub
cp /usr/share/grub/i386-pc/stage{1,2} /boot/grub
```

Remplacez `i386-pc` avec le répertoire approprié pour votre matériel.

Le répertoire `i386-pc` contient aussi un certain nombre de fichiers `*stage1_5`, différents suivant les systèmes de fichiers. Jetez un oeil sur ceux disponibles et copiez les plus appropriés dans le répertoire `/boot/grub`. La plupart des personnes copieront les fichiers `e2fs_stage1_5` et/ou `reiserfs_stage1_5`.

Installer Gzip-1.3.5

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  2,6 Mo
```

Contenu de Gzip

Le package Gzip contient des programmes pour compresser et décompresser des fichiers utilisant le codage Lempel-Ziv (LZ77).

Programmes installés : gunzip (lien vers gzip), gzexe, gzip, uncompress (lien vers gunzip), zcat (lien vers gzip), zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore et znew

Dépendances d'installation de Gzip

Gzip dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation de Gzip

Préparez la compilation de Gzip :

```
./configure --prefix=/usr
```

Le programme **`gzexe`** dispose de l'emplacement de **`gzip`** codé en dur. Comme nous changerons plus tard l'emplacement de ce binaire, la commande suivante nous assure que le nouvel emplacement sera placé dans le binaire :

```
cp gzexe.in{,.backup}
sed 's%"BINDIR"%/bin%' gzexe.in.backup > gzexe.in
```

Compilez le paquet :

```
make
```

Installez le paquet :

```
make install
```

Et déplacez les programmes dans le répertoire `/bin` :

```
mv /usr/bin/gzip /bin
rm /usr/bin/{gunzip,zcat}
ln −s gzip /bin/gunzip
ln −s gzip /bin/zcat
ln −s gunzip /bin/uncompress
```

Installer Man−1.5m2

```
Estimation du temps de constructionckage :           0,1 SBU
Estimation de l'espace disque requisckage :  1,9 Mo
```

Contenu de Man

Man est un afficheur de pages man.

Programmes installés : apropos, makewhatis, man, man2dvi, man2html et whatis

Dépendances d'installation de Man

Man dépend de Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Installation de Man

Nous allons faire quelques ajustements aux sources de Man.

Le premier correctif met en commentaire la ligne "MANPATH /usr/man" du fichier `man.conf` pour empêcher des résultats redondants lors de l'utilisation de programmes comme **`whatis`** :

```
patch −Np1 −i ../man−1.5m2−manpath.patch
```

Le second correctif ajoute l'option *−R* à la variable *PAGER* de façon à ce que les séquences d'échappement soient correctement gérées :

```
patch −Np1 −i ../man−1.5m2−pager.patch
```

Le troisième et dernier correctif supprime un problème avec les pages man formatées sur plus de 80 colonnes

utilisées avec les dernières versions de **groff**:

```
patch -Np1 -i ../man-1.5m2-80cols.patch
```

Maintenant, préparez la compilation de Man :

```
./configure -default -confdir=/etc
```

La signification des options de configure est la suivante :

- **-default** : Ceci indique au script configure de sélectionner un ensemble intéressant d'options par défaut. Par exemple, seulement les pages man anglaises, pas de catalogue de messages, man sans suid, gestion des pages man compressés, compresser les pages cat, créer les pages cat lorsque le répertoire approprié existe, suivre FHS en plaçant les pages cat sous /var/cache/man en supposant que ce répertoire existe.
- **-confdir=/etc**: Ceci indique au programme **man** de chercher le fichier de configuration `man.conf` dans le répertoire `/etc`.

Compilez le paquet :

```
make
```

Et installez-le :

```
make install
```

> **Note :** Si vous souhaitez désactiver les séquences d'échappement SGR, vous devriez éditer le fichier man.conf et ajouter l'argument **-c** pour nroff.

Vous pourriez vouloir aussi jeter un oeil à la page BLFS http://www.linuxfromscratch.org/blfs/view/cvs/postlfs/compressdoc.html gérant les problèmes de formatage et de compression pour les pages man.

Installer Make-3.80

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  8,8 Mo
```

Contenu de Make

Make détermine, automatiquement, quelles pièces d'un grand programme ont besoin d'être recompilées et envoie les commandes pour les recompiler.

Programme installé : make

Dépendances d'installation de Make

Make dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Sed.

Installation de Make

Préparez la compilation de Make :

```
./configure --prefix=/usr
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Enfin, installez le paquet :

```
make install
```

Installer Modutils−2.4.25

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  2,9 Mo
```

Contenu de Modutils

Le package Modutils contient des programmes que vous pouvez utiliser pour gérer les modules du noyau.

Programmes installés : depmod, genksyms, insmod, insmod_ksymoops_clean, kallsyms (lien vers insmod), kernelversion, ksyms (lien vers insmod), lsmod (lien vers insmod), modinfo, modprobe (lien vers insmod) et rmmod (lien vers insmod)

Dépendances d'installation de Modutils

Modutils dépend de Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Glibc, Grep, M4, Make, Sed.

Installation de Modutils

Préparez la compilation de Modutils :

```
./configure
```

Compilez le paquet :

```
make
```

Et installez−le :

```
make install
```

Installer Patch−2.5.4

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  1,9 Mo
```

Contenu de Patch

Le programme patch modifie un fichier suivant un fichier patch, appelé aussi correctif. Un correctif est habituellement une liste, créée par le programme diff, et contenant les instructions sur la façon dont le fichier original a été modifié.

Programme installé : patch

Dépendances d'installation de Patch

Patch dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Installation de Patch

Préparez la compilation de Patch :

```
CPPFLAGS=-D_GNU_SOURCE ./configure --prefix=/usr
```

De nouveau, l'option du préprocesseur **`-D_GNU_SOURCE`** est uniquement nécessaire sur la plateforme PowerPC. Sur les autres architectures, vous pouvez l'oublier.

Compilez le paquet :

```
make
```

Et installez−le :

```
make install
```

Installer Procinfo−18

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  0,2 Mo
```

Contenu de Procinfo

Le programme procinfo récupère les données système, telles que l'usage de la mémoire et les numéros d'IRQ, à partir du répertoire `/proc` et formatte ces données d'une façon conséquente.

Programmes installés : lsdev, procinfo et socklist

Dépendances d'installation de Procinfo

Procinfo dépend de Binutils, GCC, Glibc, Make, Ncurses.

Installation de Procinfo

Compilez Procinfo :

```
make LDLIBS=-lncurses
```

La signification du paramètre de make est :

- **`LDLIBS=-lncurses`** : Ceci indique à Procinfo d'utiliser la bibliothèque `libncurses` au lieu de `libtermcap`, obsolète depuis longtemps.

Enfin, installez le paquet :

```
make install
```

Installer Procps-3.1.11

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  6,2 Mo
```

Contenu de Procps

Le package Procps apporte les programmes pour observer et arrêter les processus systèmes. Procps récupère des informations sur les processus via le répertoire `/proc`.

Programmes installés : free, kill, pgrep, pkill, pmap, ps, skill, snice, sysctl, tload, top, uptime, vmstat, w et watch

Bibliothèque installée : libproc.so

Dépendances d'installation de Procps

Procps dépend de Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses.

Installation de Procps

Tout d'abord, corrigez un problème pouvant faire quitter brutalement **`w`** avec certains paramètrages de locale :

```
patch -Np1 -i ../procps-3.1.11-locale-fix.patch
```

Maintenant, compilez Procps :

```
make
```

Installez−le :

```
make install
```

Et supprimer un lien vers une bibliothèque :

```
rm /lib/libproc.so
```

Installer Psmisc−21.3

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  2,2 Mo
```

Contenu de Psmisc

Le package Psmisc contient trois programmes qui aident à gérer le répertoire `/proc`.

Programmes installés : fuser, killall et pstree

Dépendances d'installation de Psmisc

Psmisc dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed.

Installation de Psmisc

Préparez la compilation de Psmisc :

```
./configure --prefix=/usr --exec-prefix=/
```

La signification de l'option de configure est :

- **`--exec-prefix=/`** : Ceci fait que les binaires sont installés dans `/bin` au lieu de `/usr/bin`. Comme les programmes Psmisc sont souvent utilisés dans les scripts de démarrage, ils se doivent d'être disponibles aussi lorsque le système de fichiers `/usr` n'est pas monté.

Compilez le paquet :

```
make
```

Et installez−le :

```
make install
```

Par défaut, le programme **`pidof`** de Psmisc n'est pas installé. Généralement, ce n'est pas un problème car nous installons plus tard le package Sysvinit qui apporte un programme **`pidof`** supérieur. Mais si vous ne comptez pas utiliser Sysvinit, vous devez compléter l'installation de Pscmisc en créant le lien symbolique suivant :

```
ln -s killall /bin/pidof
```

Installer Shadow−4.0.3

```
Estimation du temps de construction :           0,4 SBU
Estimation de l'espace disque requis :  11 Mo
```

Contenu de Shadow

Le package Shadow a été créé pour augmenter la sécurité des mots de passe du système.

Programmes installés : chage, chfn, chpasswd, chsh, dpasswd, expiry, faillog, gpasswd, groupadd, groupdel, groupmod, groups, grpck, grpconv, grpunconv, lastlog, login, logoutd, mkpasswd, newgrp, newusers, passwd, pwck, pwconv, pwunconv, sg (lien vers newgrp), useradd, userdel, usermod, vigr (lien vers vipw) et vipw

Dépendances d'installation de Shadow

Shadow dépend de Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Installation de Shadow

Les programmes **`login`**, **`getty`** et **`init`** (ainsi que quelques autres) maintiennent des journaux pour enregistrer qui est ou était connecté au système. Néanmoins, ces programmes ne créent pas ces journaux lorsque ceux−ci n'existent pas. Donc, si vous souhaitez que ces traces soient générées, vous devrez créer ces fichiers vous−même. Le package Shadow doit détecter ces fichiers au bon endroit, donc nous les créons maintenant avec de bons droits :

```
touch /var/run/utmp /var/log/{btmp,lastlog,wtmp}
chmod 644 /var/run/utmp /var/log/{btmp,lastlog,wtmp}
```

Le fichier `/var/run/utmp` contient la liste des utilisateurs actuellement connectés, le fichier `/var/log/wtmp` ceux qui l'*étaient* et quand. Le fichier `/var/log/lastlog` affiche pour chaque utilisateur quand il, ou elle, s'est connecté la dernière fois alors que le fichier `/var/log/btmp` contient les tentatives échouées de connexion.

Shadow code en dur le chemin vers le binaire **`passwd`** à l'intérieur du binaire lui−même mais fait cela d'une mauvaise façon. Si un binaire **`passwd`** n'est pas présent lors de l'installation de Shadow, le package suppose de manière erronée que celui−ci se trouve dans `/bin` mais l'installe ensuite dans `/usr/bin`. Ceci amène des erreurs sur l'absence de `/bin/passwd`. Pour contourner ce problème, créez un fichier `passwd` de façon à ce qu'il soit bien codé :

```
touch /usr/bin/passwd
```

La suite actuelle Shadow a un problème avec la commande **`newgrp`** faisant échouer celle−ci. Le correctif suivant (apparaissant aussi dans le code CVS de Shadow) corrige ce problème :

```
patch -Np1 -i ../shadow-4.0.3-newgrp-fix.patch
```

Maintenant, préparez la compilation de Shadow :

```
./configure --prefix=/usr --libdir=/usr/lib --enable-shared
```

Compilez le paquet :

```
make
```

Et installez-le :

```
make install
```

Shadow utilise deux fichiers pour configurer les paramétrages d'authentification du système. Installez ces deux fichiers de configuration :

```
cp etc/{limits,login.access} /etc
```

Nous voulons modifier la méthode de cryptage du mot de passe en activant les mots de passe MD5 théoriquement plus sécurisés qu'avec la méthode par défaut, crypt, et permettant en plus des mots de passe dépassant huit caractères. Nous avons aussi besoin de changer l'ancien emplacement `/var/spool/mail` des boîtes mail des utilisateurs par l'emplacement actuel, `/var/mail`. Nous faisons cela en modifiant le fichier de configuration correct lors de sa copie :

```
sed -e 's%/var/spool/mail%/var/mail%' \
    -e 's%#MD5_CRYPT_ENAB.no%MD5_CRYPT_ENAB yes%' \
    etc/login.defs.linux > /etc/login.defs
```

> **Note :** Faites attention en tapant la commande ci-dessus. Il est probablement plus sûr de la copier et coller plutôt que d'essayer de la taper entièrement.

En accord avec la page man de **`vipw`**, un programme **`vigr`** devrait aussi exister. Comme la procédure d'installation ne crée pas ce programme, créons le lien symbolique manuellement :

```
ln -s vipw /usr/sbin/vigr
```

Comme le lien symbolique `/bin/vipw` est redondant (et pointe même vers un fichier inexistant), supprimez-le :

```
rm /bin/vipw
```

Maintenant, déplacez le programme **`sg`** à son bon emplacement :

```
mv /bin/sg /usr/bin
```

Et déplacez les bibliothèques partagées de Shadow à un endroit plus approprié :

```
mv /usr/lib/lib{shadow,misc}.so.0* /lib
```

Comme certains packages s'attendent à trouver les bibliothèques tout juste déplacées dans `/usr/lib`, créez les liens symboliques suivants :

```
ln -sf ../../lib/libshadow.so.0 /usr/lib/libshadow.so
ln -sf ../../lib/libmisc.so.0 /usr/lib/libmisc.so
```

Coreutils a déjà installé un programme **groups** dans /usr/bin. Si vous le souhaitez, vous pouvez supprimer celui installé par Shadow :

```
rm /bin/groups
```

Configurer Shadow

Ce paquet contient des utilitaires pour modifier les mots de passe des utilisateurs, ajouter ou supprimer des utilisateurs et groupes, et plus encore. Nous n'allons pas expliquer ce que signifie 'shadow password'. Une explication complète peut être lue dans le fichier doc/HOWTO compris dans l'arborescence des sources du paquet Shadow. Il y a une chose que vous devez garder à l'esprit, si vous décidez d'activer le support de Shadow : les programmes qui ont besoin de vérifier des mots de passe (comme xdm, les démons ftp, pop3, etc) doivent être compatibles avec Shadow, c'est−à−dire qu'ils doivent être capables de travailler avec des mots de passe Shadow.

Pour activer les mots de passe shadow, lancez la commande suivante :

```
/usr/sbin/pwconv
```

Et pour activer les mots de passe shadow pour les groupes, lancez la commande suivante :

```
/usr/sbin/grpconv
```

Sous des circonstances normales, vous n'avez pas encore créé de mots de passe. Néanmoins, si vous retournez dans cette section pour activer les mots de passe Shadow, vous devriez réinitialiser les mots de passe de tous les utilisateurs avec la commande **passwd** ainsi que les mots de passe de tous les groupes en utilisant la commande **gpasswd**.

Installer Sysklogd−1.4.1

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  0,5 Mo
```

Contenu de Sysklogd

Le package Sysklogd contient des programmes pour enregistrer les messages de trace du système, tels que ceux donnés par le noyau.

Programmes installés : klogd et syslogd

Dépendances d'installation de Sysklogd

Sysklogd dépend de Binutils, Coreutils, GCC, Glibc, Make.

Installation de Sysklogd

Compilez Sysklogd :

```
make
```

Et installez−le :

```
make install
```

Configurer Sysklogd

Créez un nouveau fichier `/etc/syslog.conf` en exécutant ce qui suit :

```
cat > /etc/syslog.conf << "EOF"
# Debut de /etc/syslog.conf

auth,authpriv.* −/var/log/auth.log
*.*;auth,authpriv.none −/var/log/sys.log
daemon.* −/var/log/daemon.log
kern.* −/var/log/kern.log
mail.* −/var/log/mail.log
user.* −/var/log/user.log
*.emerg *

# Fin de /etc/syslog.conf
EOF
```

Installer Sysvinit−2.85

```
Estimation du temps de construction :           0,1 SBU
Estimation de l'espace disque requis :  0,9 Mo
```

Contenu de Sysvinit

Le package Sysvinit contient des programmes pour contrôler le démarrage, l'exécution et l'arrêt de tous les autres programmes.

Programmes installés : halt, init, killall5, last, lastb (lien vers last), mesg, pidof (lien vers killall5), poweroff (lien vers halt), reboot (lien vers halt), runlevel, shutdown, sulogin, telinit (lien vers init), utmpdump et wall

Dépendances d'installation de Sysvinit

Sysvinit dépend de Binutils, Coreutils, GCC, Glibc, Make.

Installation de Sysvinit

Lorsque les niveaux d'exécution sont modifiés (par exemple, lors de l'arrêt du système), init envoie les signaux TERM et KILL aux processus qu'il a lancé. Init affiche "Sending processes the TERM signal" sur

l'écran. Ceci semble impliquer qu'init envoie ces signaux à tous les processus en cours d'exécution. Pour éviter la confusion, le fichier init.c va être modifié de façon à ce que le message soit "Sending processes started by init the TERM signal".

Editez le message d'arrêt :

```
cp src/init.c{,.backup}
sed 's/Sending processes/Sending processes started by init/g' \
    src/init.c.backup > src/init.c
```

Compilez Sysvinit :

```
make −C src
```

Et installez−le :

```
make −C src install
```

Configurer Sysvinit

Créez un nouveau fichier `/etc/inittab` en exécutant ceci :

```
cat > /etc/inittab << "EOF"
# Debut de /etc/inittab

id:3:initdefault:

si::sysinit:/etc/rc.d/init.d/rc sysinit

l0:0:wait:/etc/rc.d/init.d/rc 0
l1:S1:wait:/etc/rc.d/init.d/rc 1
l2:2:wait:/etc/rc.d/init.d/rc 2
l3:3:wait:/etc/rc.d/init.d/rc 3
l4:4:wait:/etc/rc.d/init.d/rc 4
l5:5:wait:/etc/rc.d/init.d/rc 5
l6:6:wait:/etc/rc.d/init.d/rc 6

ca:12345:ctrlaltdel:/sbin/shutdown −t1 −a −r now

su:S016:once:/sbin/sulogin

1:2345:respawn:/sbin/agetty tty1 9600
2:2345:respawn:/sbin/agetty tty2 9600
3:2345:respawn:/sbin/agetty tty3 9600
4:2345:respawn:/sbin/agetty tty4 9600
5:2345:respawn:/sbin/agetty tty5 9600
6:2345:respawn:/sbin/agetty tty6 9600

# Fin de /etc/inittab
EOF
```

Installer Tar−1.13.25

```
Estimation du temps de construction :           0,2 SBU
```

```
Estimation de l'espace disque requis :  10 Mo
```

Contenu de Tar

Tar est un programme d'archivage permettant de stocker et d'extraire des fichiers à partir d'un fichier d'archive connu comme un fichier tar.

Programmes installés : rmt et tar

Dépendances d'installation de Tar

Tar dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Installation de Tar

Préparez la compilation de Tar :

```
./configure --prefix=/usr --bindir=/bin \
    --libexecdir=/usr/bin
```

Compilez le paquet :

```
make
```

Ce paquet dispose d'une suite de tests, réalisant plusieurs tests pour s'assurer que le programme a été construit correctement. Si vous décidez de le lancer, la commande suivante fera le nécessaire :

```
make check
```

Enfin, installez le paquet :

```
make install
```

Installer Util−linux−2.12

```
Estimation du temps de construction :           0,2 SBU
Estimation de l'espace disque requis :  16 Mo
```

Contenu de Util−linux

Le package Util−linux contient un certain nombre d'utilitaires divers. Certains des plus connus d'entre eux sont utilisés pour monter, démonter, formatter, partionner et gérer des disques durs, ouvrir des ports tty ainsi que récupérer des messages du noyau.

Programmes installés : agetty, arch, blockdev, cal, cfdisk, chkdupexe, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, dmesg, elvtune, fdformat, fdisk, fsck.cramfs, fsck.minix, getopt, hexdump, hwclock, ipcrm, ipcs, isosize, kill, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, parse.bash, parse.tcsh, pg, pivot_root, ramsize (lien vers rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (lien vers rdev), script, setfdprm, setsid, setterm, sfdisk, swapoff (lien vers

swapon), swapon, test.bash, test.tcsh, tunelp, ul, umount, vidmode (lien vers rdev), whereis et write

Dépendances d'installation de Util−linux

Util−linux dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Zlib.

Notes de compatibilité FHS

Le FHS recommande d'utiliser `/var/lib/hwclock`, à la place de l'habituel `/etc`, comme emplacement du fichier `adjtime`. Pour rendre le programme **`hwclock`** compatible avec FHS, lancez ce qui suit :

```
cp hwclock/hwclock.c{,.backup}
sed 's%etc/adjtime%var/lib/hwclock/adjtime%' \
    hwclock/hwclock.c.backup > hwclock/hwclock.c
mkdir −p /var/lib/hwclock
```

Installation de Util−linux

Préparez la compilation de Util−linux :

```
./configure
```

Compilez le paquet :

```
make HAVE_SLN=yes
```

La signification du paramètre de make est :

- **`HAVE_SLN=yes`** : Ceci empêche le programme **`sln`** (une version liée statiquement de **`ln`**, déjà installé par Glibc) d'être de nouveau construit.

Et installez le paquet :

```
make HAVE_SLN=yes install
```

Installer GCC−2.95.3

```
Estimation du temps de construction :    1,5 SBU
Estimation de l'espace disque requis :  130 Mo
```

Installation de GCC

Ce paquet est connu pour mal se comporter si vous changez les options d'optimisation par défaut (en incluant les options −march et −mcpu). Donc, si vous avez défini des variables d'environnement qui surchargent les optimisations pas défaut, telles que CFLAGS et CXXFLAGS, nous vous recommandons de supprimer cette initialisation lors de la construction de GCC.

C'est une ancienne version de GCC que nous allons installer pour compiler le noyau Linux au chapitre 8. Cette version est recommandée par les développeurs du noyau lorsque vous avez besoin d'une stabilité absolue. Les versions suivantes de GCC n'ont pas été autant testées pour la compilation du noyau Linux. Utiliser une de ces versions a de bonne chance de fonctionner, néanmoins, nous vous recommandons d'adhérer au conseil des développeurs du noyau et d'utiliser cette version pour compiler votre noyau.

> **Note :** Nous n'installons pas le compilateur C++ ou les bibliothèques ici. Néanmoins, vous pouvez avoir de bonnes raisons pour souhaiter les installer. Vous trouverez plus d'informations sur http://www.linuxfromscratch.org/blfs/view/stable/general/gcc2.html.

Nous installerons cette ancienne version de GCC avec comme préfixe, non standard, `/opt` de façon à éviter les interférences avec le système GCC déjà installé dans `/usr`.

Appliquez les correctifs et faites un petit ajustement :

```
patch -Np1 -i ../gcc-2.95.3-2.patch
patch -Np1 -i ../gcc-2.95.3-no-fixinc.patch
patch -Np1 -i ../gcc-2.95.3-returntype-fix.patch
echo timestamp > gcc/cstamp-h.in
```

La documentation GCC recommande la construction de GCC en dehors du répertoire des sources, c'est-à-dire dans un répertoire dédié :

```
mkdir ../gcc-2-build
cd ../gcc-2-build
```

Compilez et installez le compilateur :

```
../gcc-2.95.3/configure --prefix=/opt/gcc-2.95.3 \
    --enable-shared --enable-languages=c \
    --enable-threads=posix
make bootstrap
make install
```

Commande chroot revue

A partir de maintenant, lorsque vous sortez de l'environnement chroot et que vous souhaitez y retourner, vous devez lancer la commande chroot suivante :

```
chroot $LFS /usr/bin/env -i \
    HOME=/root TERM=$TERM PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin \
    /bin/bash --login
```

La raison est que nous n'avons plus besoin d'utiliser les programmes à partir du répertoire `/tools`. Néanmoins, nous ne voulons pas encore supprimer le répertoire `/tools`. Il aura encore son utilité jusqu'à la fin de ce livre.

Installer LFS-Bootscripts-1.12

```
Estimation du temps de compilation :           0,1 SBU
Estimation de l'espace disque requis :  0,3 Mo
```

Contenu de LFS−bootscripts

Le package LFS−Bootscripts contient des scripts shell style init SysV. Ces scripts réalisent plusieurs tâches telles que la vérification de l'intégrité d'un système de fichiers lors du redémarrage, le chargement du plan de codage du clavier, la configuration du réseau et l'arrêt des processus lors de l'arrêt du système.

Scripts installés : checkfs, cleanfs, functions, halt, ifdown, ifup, loadkeys, localnet, mountfs, mountproc, network, rc, reboot, sendsignals, setclock, swap, sysklogd et template

Dépendances d'installation de Bootscripts

Bzip2 dépend de Bash, Coreutils.

Installation de LFS−Bootscripts

Nous allons utiliser les scripts de démarrage SysV. Nous avons choisi ce style parce qu'il est communément utilisé et que nous le connaissons bien. Si vous préférez essayer autre chose, Marc Heerdink a écrit une astuce sur les scripts d'initialisation style BSD, disponible sur http://www.linuxfromscratch.org/hints/downloads/files/bsd−init.txt. Et si vous souhaitez quelque chose de plus radical, recherchez depinit dans les listes de diffusion LFS.

Si vous décidez d'utiliser le style BSD, ou tout autre style de scripts, vous pouvez passer le chapitre suivant lorsque vous y arriverez et aller directement au chapitre 8.

Installez les scripts de démarrage :

```
cp −a rc.d sysconfig /etc
```

Faites que *root* soit propriétaire des scripts :

```
chown −R root:root /etc/rc.d /etc/sysconfig
```

Configurer les composants du système

Maintenant que tous les logiciels sont installés, tout ce qui nous reste à faire est de réaliser quelques configurations.

Configuration du clavier

Il y a des choses plus ennuyeuses que d'utiliser Linux avec un mauvais plan de codage pour votre clavier. Si vous disposez du clavier US standard, vous pouvez passer cette section car le plan de clavier US est celui par défaut tant que celui−ci n'est pas modifié.

Pour modifier le fichier de plan de codage par défaut, créez le lien symbolique `/usr/share/kbd/keymaps/defkeymap.map.gz` en lançant la commande suivante :

```
ln −s path/to/keymap /usr/share/kbd/keymaps/defkeymap.map.gz
```

Bien sûr, remplacez `chemin/vers/keymap` avec le chemin et le nom du fichier pour votre clavier. Par exemple, si vous utilisez un clavier hollandais, vous utiliserez `i386/qwerty/nl.map.gz`.

Une autre façon pour configurer votre plan de codage de votre clavier est de compiler celui−ci dans le noyau. Ceci vous assurera que votre clavier fonctionnera toujours correctement, même lorsque vous démarrez en mode maintenance (en passant 'init=/bin/sh' au noyau), mode qui ne lance pas le script de démarrage qui configure votre clavier.

Lancer la commande suivante pour corriger le bon plan de codage dans les sources du noyau. Vous devrez répéter cette commande chaque fois que vous déballerez un nouveau noyau :

```
loadkeys −m /usr/share/kbd/keymaps/defkeymap.map.gz > \
    /usr/src/linux−2.4.22/drivers/char/defkeymap.c
```

Ajouter un mot de passe pour root

Choisissez un mot de passe pour l'utilisateur root et créez−le en lançant la commande suivante :

```
passwd root
```

Chapitre 7. Mise en place des scripts de démarrage

Introduction

Ce chapitre va permettre la configuration des scripts de démarrage que vous avez installés lors du chapitre 6. La plupart de ces scripts fonctionneront sans modification, mais un petit nombre vont nécessiter le paramétrage de fichiers de configurations supplémentaires, car ils ont affaire avec des informations dépendantes du matériel.

Comment fonctionne le processus de démarrage utilisant ces scripts?

Linux utilise un système de démarrage nommé SysVinit. Il est basé sur le concept de *niveaux d'exécution*. Il peut être extrêmement différent d'un système à l'autre et on ne peut donc pas supposer que si cela a fonctionné avec <telle distribution>, cela devrait aussi fonctionner de la même manière avec LFS. LFS fait les choses à sa manière, mais respecte généralement les standards établis.

SysVinit (que nous nommerons dorénavant *init*) se base pour fonctionner sur un système de niveaux d'exécution. Ils sont au nombre de 7 (de 0 à 6) (en réalité, il y a plus de niveaux d'exécution que cela, mais ils sont réservés à des cas spéciaux et ne sont généralement pas utilisés. La page man de init décrit ces détails), et chacun d'eux correspond à ce que l'ordinateur est supposé faire lorsqu'il démarre. Le niveau d'exécution par défaut est le niveau 3. Voici la description des différents niveaux d'exécution tels qu'ils sont fréquemment implémentés :

0: arrête l'ordinateur ; 1: mode mono−utilisateur ; 2: mode multi−utilisateur sans réseau ;
3: mode multi−utilisateur avec le réseau ; 4: réservé à la personnalisation, sinon identique au niveau 3 ;
5: identique au 4, utilisé généralement pour une connexion graphique (comme l'xdm de X ou kdm de KDE) ;
6: redémarre l'ordinateur.

L'instruction utilisée pour changer de niveau d'exécution est **`init <niveau d'exécution>`** où <niveau d'exécution> est le niveau d'exécution désiré. Par exemple, pour redémarrer l'ordinateur, un utilisateur lancera l'instruction init 6. L'instruction reboot est simplement un alias, tout comme l'instruction halt est un alias pour init 0.

Il existe un certain nombre de répertoires sous /etc qui ressemblent à rc?.d où ? est le niveau d'exécution et rcsysinit.d qui contient un certain nombre de liens symboliques. Certains commencent par un K, les autres par un S et tous ont deux chiffres après la lettre initiale. Le K signifie d'arrêter (kill) un service, et le S (start) d'en démarrer un. Les chiffres déterminent l'ordre d'exécution des scripts, de 00 à 99 ; plus un nombre est petit, plus tôt il sera exécuté. Lorsque init passe à un autre niveau d'exécution, les services appropriés sont arrêtés et d'autres sont démarrés.

Les véritables scripts sont dans /etc/rc.d/init.d. Ils font tout le travail et les liens symboliques pointent vers eux. Les liens d'arrêt et de démarrage pointent vers le même script dans /etc/rc.d/init.d. Ceci est dû au fait que le script peut être appelé avec des paramètres différents tels que start, stop, restart, reload, status. Quand un lien K est rencontré, le script approprié est exécuté avec l'argument stop. Quand un lien S est rencontré, le script idoine est exécuté avec l'argument start.

Il existe une exception. les liens commençant par un S dans les répertoires rc0.d et rc6.d ne feront rien

démarrer. Ils seront appelés avec le paramètre *stop* pour arrêter quelque chose. La logique derrière cela est que lorsque l'on est sur le point de redémarrer ou d'arrêter le système, on ne désire rien démarrer, seulement arrêter le système.

Voici la description de ce que ces arguments font réaliser aux scripts :

- *start* : le service est démarré ;
- *stop* : le service est arrêté ;
- *restart* : le service est arrêté puis démarré à nouveau ;
- *reload* : la configuration du service est mise à jour. Ceci est utilisé après la modification du fichier de configuration d'un service, lorsque le service n'a pas besoin d'être redémarré ;
- *status* : indique si le service est lancé et avec quel PID.

Modifiez librement la manière dont le système de démarrage fonctionne (après tout, il s'agit de votre système LFS). Les fichiers donnés ici sont simplement des exemples de la manière de faire tout cela proprement (quoi qu'il en soit, vous pouvez détester ce que nous considérons comme propre).

Configuration du script setclock

Le script setclock lit l'heure à partir de l'horloge matérielle (aussi connu en tant qu'horloge BIOS ou CMOS) et soit convertit cette heure en temps local en utilisant le fichier `/etc/localtime` (si l'horloge matérielle est réglée sur GMT), soit ne le fait pas (si votre horloge matérielle est déjà réglé sur l'heure locale). Il n'existe pas de moyens pour détecter automatiquement si votre horloge est réglée sur l'heure GMT ou locale, donc nous avons besoin de la configurer nous−même.

Changer la valeur de la variable *UTC* ci−dessous en indiquant *0* (zéro) si votre horloge matérielle n'utilise pas l'heure GMT.

Créez un nouveau fichier `/etc/sysconfig/clock` en lançant la commande suivante :

```
cat > /etc/sysconfig/clock << "EOF"
# Debut /etc/sysconfig/clock

UTC=1

# Fin /etc/sysconfig/clock
EOF
```

Maintenant, vous voudrez peut−être jeter un oeil sur cette excellente astuce expliquant comment on gère le temps avec LFS : http://www.linuxfromscratch.org/hints/downloads/files/time.txt. Elle explique des points tels que les zones horaires, UTC, et la variable d'environnement TZ.

Ai−je besoin du script loadkeys?

Si vous avez décidé de compiler votre plan de codage du clavier directement dans le noyau à la fin du chapitre 6, alors vous n'avez pas besoin de lancer le script loadkeys car le noyau a déjà initialisé le plan de codage pour vous. Vous pouvez toujours le lancer si vous voulez, cela ne risque pas de vous poser problème. Cela peut même être avantageux de le garder au cas où vous posséderiez différents noyaux et que vous ne vous rappelleriez pas, ou que vous ne voudriez pas compiler le plan de codage dans chaque noyau installé.

Si vous décidez que vous n'en avez pas besoin ou que vous ne voulez pas utiliser le script loadkeys, supprimez le lien symbolique `/etc/rc.d/rcsysinit.d/S70loadkeys`.

Configuration du script sysklogd

Le script `sysklogd` invoque le programme **`syslogd`** avec l'option *−m 0*. Cette option désactive la marque périodique que syslogd écrit sur les journaux système toutes les vingt minutes par défaut. Si vous préférez l'activer, éditez le script `sysklogd` et faites les changements adéquats. Voir la page de manuel **`man syslogd`** pour plus d'informations.

Configurer le script localnet

Une partie de ce script configure le nom du système. Ce nom doit être indiqué dans le fichier /etc/sysconfig/network.

Créez le fichier /etc/sysconfig/network et entrez le nom du système en lançant :

```
echo "HOSTNAME=lfs" > /etc/sysconfig/network
```

<< lfs >> doit être remplacé par le nom de l'ordinateur. Vous ne devez pas entrer le FQDN (Fully Qualified Domain Name, nom de domaine pleinement qualifié) ici. Cette information sera entrée dans le fichier `/etc/hosts` un peu plus tard.

Créer le fichier /etc/hosts

Si une carte réseau doit être configurée, vous devez choisir l'adresse IP, le nom de domaine pleinement qualifié et les alias possibles à déclarer dans le fichier /etc/hosts. La syntaxe est la suivante :

```
<adresse IP> mon-hôte.mon-domaine.org aliases
```

Vous devez vous assurer que l'adresse IP se trouve dans la plage d'adresses réservée aux réseaux privés. Les plages valides sont :

```
        Classes Réseaux
        A     10.0.0.0
        B     172.16.0.0 à 172.31.0.0
        C     192.168.0.0 à 192.168.255.0
```

Une adresse IP valide pourrait être 192.168.1.1. Un nom de domaine pleinement qualifié pour cette adresse IP pourrait être www.linuxfromscratch.org.

Si vous ne possédez pas de carte réseau, vous devez néanmoins déclarer un nom de domaine pleinement qualifié. Cela est nécessaire à certains programmes pour fonctionner correctement.

Si aucune carte réseau ne doit être configurée, créez le fichier `/etc/hosts` en lançant la commande :

```
cat > /etc/hosts << "EOF"
# Début de /etc/hosts (version sans carte réseau)

127.0.0.1 www.mon-domaine.com <nom d'hôte> localhost
```

```
# Fin de /etc/hosts (version sans carte réseau)
EOF
```

Si une carte réseau doit être configurée, créez le fichier `/etc/hosts` en lançant la commande :

```
cat > /etc/hosts << "EOF"
# Début de /etc/hosts (version avec carte réseau)

127.0.0.1 localhost.localdomain localhost
192.168.1.1 www.mon-domaine.org <nom d'hôte>

# Fin de /etc/hosts (version avec carte réseau)
EOF
```

Bien évidemment, les valeurs 192.168.1.1 et www.mon−domaine.org doivent être changées selon vos souhaits (ou selon les valeurs données par l'administrateur système/réseau si cette machine doit être connectée à un réseau existant).

Configuration du script network

Cette section n'est utile que si vous souhaitez configurer une carte réseau.

Si vous ne possédez aucune carte réseau, vous pouvez ne pas créer les fichiers de configuration en rapport avec celle−ci. Si c'est le cas, vous devez supprimer les liens symboliques de `network` de toutes les répertoires des niveaux d'exécution (`/etc/rc.d/rc*.d`)

Configuration de la passerelle par défaut

Si vous êtes sur un réseau, vous pouvez avoir besoin de configurer la passerelle par défaut de cette machine. Ceci se fait en ajoutant les bonnes valeurs au fichier /etc/sysconfig/network, en lançant les commandes suivantes :

```
cat >> /etc/sysconfig/network << "EOF"
GATEWAY=192.168.1.2
GATEWAY_IF=eth0
EOF
```

Les valeurs pour GATEWAY et GATEWAY_IF doivent être changées pour correspondre à la configuration de votre réseau. GATEWAY contient l'adresse IP de la passerelle par défaut, et GATEWAY_IF contient l'interface réseau par laquelle la passerelle par défaut peut être jointe.

Création des fichiers de configuration d'interfaces

Les interfaces qui doivent être activées ou non dépendent des fichiers du répertoire /etc/sysconfig/network−devices. Ce répertoire contient des fichiers de la forme ifconfig.xyz, où xyz est le nom de l'interface (comme eth0 ou eth0:1).

Si vous décidez de renommer ou changer le répertoire /etc/sysconfig/network−devices, assurez−vous de modifier le fichier /etc/sysconfig/rc et de mettre à jour la variable network_devices en lui indiquant le nouveau chemin.

Maintenant, de nouveaux fichiers sont créés dans ce répertoire avec les informations ci−dessous. La commande suivante crée un fichier d'exemple ifconfig.eth0 :

```
cat > /etc/sysconfig/network−devices/ifconfig.eth0 << "EOF"
ONBOOT=yes
IP=192.168.1.1
NETMASK=255.255.255.0
BROADCAST=192.168.1.255
EOF
```

Bien sûr, les valeurs de ces variables doivent être changées dans chaque fichier pour correspondre à votre propre configuration. Si la variable ONBOOT est mise à yes, le script network l'activera lors du démarrage du système. Pour une quelconque autre valeur, cette interface sera ignorée par le script network et donc non activée.

Chapitre 8. Rendre le système LFS démarrable

Introduction

Ce chapitre permettra de rendre LFS démarrable. Ce chapitre traite de la création d'un nouveau fichier fstab, de la construction d'un nouveau noyau pour le nouveau système LFS et de l'installation du chargeur de démarrage Grub afin que le système LFS puisse être sélectionné au démarrage.

Créer le fichier /etc/fstab

Le fichier `/etc/fstab` est utilisé par quelques programmes pour déterminer les partitions à monter par défaut, quels systèmes de fichiers sont à vérifier et dans quel ordre. Créer un nouvelle table des systèmes de fichiers comme ceci :

```
cat > /etc/fstab << "EOF"
# Début /etc/fstab

# systeme de fichiers      type fs
#             point-de-montage      options     dump  ordre-fsck

/dev/xxx      /             fff      defaults    1     1
/dev/yyy      swap          swap     pri=1       0     0
proc          /proc         proc     defaults    0     0
shm           /dev/shm      tmpfs    defaults    0     0

# Fin /etc/fstab
EOF
```

Bien sûr, remplacez `xxx`, `yyy` et `fff` avec les valeurs appropriées pour votre système -- par exemple `hda2`, `hda5` et `reiserfs`. Pour tous les détails sur les six champs de cette table, voir **`man 5 fstab`**.

Lors de l'utilisation d'une partition reiserfs, le *1 1* à la fin de la ligne doit être remplacé par *0 0*, car une telle partition ne doit pas être dumpée ou vérifiée.

Le point de montage `/dev/shm` pour tmpfs est inclus pour permettre l'activation de la mémoire partagée POSIX. Votre noyau doit disposer du support requis en interne pour fonctionner -- plus d'informations là-dessus dans la prochaine section. Merci de noter qu'actuellement très peu de logiciels utilise la mémoire partagée POSIX. Donc, vous pouvez considérer le point de montage `/dev/shm` comme étant optionnel. Pour plus d'informations, voir `Documentation/filesystems/tmpfs.txt` dans le répertoire des sources du noyau.

Il existe d'autres lignes que vous pouvez ajouter à votre fichier `fstab`. Un exemple entre autres est la ligne que vous devez avoir pour utiliser les périphériques USB :

```
usbfs       /proc/bus/usb  usbfs    defaults    0     0
```

Cette option fonctionnera bien sûr seulement si vous avez le support compilé dans votre noyau.

Installer Linux−2.4.22

```
Temps de construction estimé:          Toutes les options par défaut : 4,20 SBU
Espace disque nécessaire estimé:  Toutes les options par défaut : 181 Mo
```

Contenu de Linux

Le noyau Linux kernel est au coeur de chaque système Linux. C'est ce qui fait tourner Linux. Lorsqu'un ordinateur est allumé et lance un système Linux, la première pièce de logiciel Linux à être chargé est le noyau. Celui−ci initialise les composants matériels du système: ports séries, ports parallèles, cartes son, cartes réseaux, contrôleurs IDE, contrôleurs SCSI et beaucoup plus encore. EN bref, le noyau rend le matériel disponible aux logiciels exécutés.

Fichiers installés : le noyau et ses en−têtes.

Dépendances d'installation de Linux

Linux dépend de Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

Installation du noyau

Construire le noyau implique un certain nombre d'étapes : configuration, compilation et installation. Si vous n'aimez pas la façon dont ce livre configure le noyau, jetez un oeil sur le fichier `README` contenu dans les sources du noyau pour d'autres méthodes.

Préparez la compilation en lançant la commande suivante :

```
make mrproper
```

Ceci nous assure que le répertoire du noyau est complètement nettoyé. L'équipe du noyau recommande que cette commande soit lancée avant *chaque* compilation du noyau. Vous ne devez pas penser que le répertoire des sources soit propre juste après avoir été déballé.

Configurez le noyau via une interface par menu :

```
make menuconfig
```

`make oldconfig` peut être plus approprié dans certaines situations. Voir le fichier `README` pour plus d'informations.

Si vous le souhaitez, vous pouvez éviter la configuration du noyau en copiant le fichier de configuration du noyau, `.config`, à partir de votre système hôte (en supposant qu'il soit disponible) vers le répertoire `$LFS/usr/src/linux−2.4.22`. Néanmoins, nous ne recommandons pas cette option. Vous ferez bien mieux d'explorer tous les menus de configuration et de créer la configuration de votre propre noyau à partir de rien.

Pour le support de la mémoire partagée POSIX, assurez−vous que l'option de configuration du noyau "Virtual memory file system support" est activée. Elle réside dans le menu "File systems" et est normalement activée

par défaut.

Il est important de noter que pour être conforme avec les réquisitions de la mémoire partagée POSIX, nous devons activer l'option du système de fichiers tmpfs et nous devons monter un système de fichiers tmpfs à partir de `/dev/shm`.

Vérifiez les dépendances et créez les fichiers d'information sur dépendances :

```
make CC=/opt/gcc-2.95.3/bin/gcc dep
```

Compilez l'image du noyau :

```
make CC=/opt/gcc-2.95.3/bin/gcc bzImage
```

Compilez les pilotes qui ont été configurés comme modules :

```
make CC=/opt/gcc-2.95.3/bin/gcc modules
```

Si vous avez l'intention d'utiliser des modules du noyau, vous aurez besoin d'un fichier `/etc/modules.conf`. L'information sur les modules et sur la configuration du noyau sont disponibles en général dans la documentation du noyau, qui est stockée dans `linux-2.4.22/Documentation`. La page man modules.conf et le HOWTO du noyau (http://www.tldp.org/HOWTO/Kernel-HOWTO.html peuvent aussi vous intéresser.

Installez les modules :

```
make CC=/opt/gcc-2.95.3/bin/gcc modules_install
```

Comme rien n'est terminé sans documentation, construisez les pages man venant avec le noyau :

```
make mandocs
```

Et installez ces pages :

```
cp -a Documentation/man /usr/share/man/man9
```

La compilation du noyau est terminée, mais certains des fichiers créés résident toujours dans le répertoire des sources. Pour terminer l'installation, deux fichiers doivent être copiés dans le répertoire `/boot`.

Le chemin vers le fichier du noyau peut varier suivant la plateforme que vous utilisez. Lancez la commande suivante pour installer le noyau :

```
cp arch/i386/boot/bzImage /boot/lfskernel
```

`System.map` est un fichier de symboles du noyau. Il indique les points d'entrées de chaque fonction dans l'API du noyau, mais aussi les adresses des structures de données propres au noyau en cours d'exécution. Lancez la commande suivante pour installer le fichier map :

```
cp System.map /boot
```

Rendre le système LFS démarrable

Votre nouveau système LFS est pratiquement fini. Une des dernières choses à faire est de vous assurer que vous pouvez le démarrer. Les instructions ci−dessous s'appliquent seulement aux ordinateurs de l'architecture IA−32, c'est−à−dire les PC standards. Des informations sur le "chargement au démarrage" pour les autres architectures devraient être disponibles aux emplacements habituelles des ressources pour ces architectures.

Le chargement au démarrage est un domaine complexe. Tout d'abord, quelques mots de mise en garde. Vous devez vraiment connaitre votre chargeur actuel et tout autre système d'exploitation présent sur votre disque dur que vous souhaitez pouvoir lancer. Assurez−vous d'avoir une disquette de démarrage de façon à pouvoir réparer votre ordinateur si, par malheur, votre ordinateur devenait inutilisable (non démarrable).

Plus tôt, nous avons compilé et installé le chargeur de démarrage Grub pour cette étape. La procédure implique l'écriture de quelques fichiers spéciaux de Grub en des endroits spécifiques sur le disque dur. Avant d'en arriver là, nous vous recommandons fortement de créer une disquette de démarrage Grub au cas où. Insérez une disquette de démarrage vierge et lancez les commandes suivantes :

```
dd if=/boot/grub/stage1 of=/dev/fd0 bs=512 count=1
dd if=/boot/grub/stage2 of=/dev/fd0 bs=512 seek=1
```

Enlevez la disquette et rangez−la dans un endroit sûr. Maintenant, nous allons lancer le shell **grub** :

```
grub
```

Grub utilise sa propre structure de nommage des disques et partitions, de la forme (hdn,m), où *n* est le numéro du disque dur et *m* le numéro de la partition, tout deux commençant à zéro. Ceci signifie, par exemple, que la partition hda1 est (hd0,0) pour Grub alors que hdb2 est (hd1,1). Contrairement à Linux, Grub ne considère pas les lecteurs de CDRoms comme des disques durs, donc si vous avez un CD sur hdb, par exemple, et un second disque dur sur hdc, ce dernier disque sera malgré tout (hd1).

En utilisant les informations ci−dessus, déterminez la désignation appropriée pour votre partition root. Pour l'exemple suivant, nous supposerons que votre partition root est hda4.

Tout d'abord, indiquez à Grub où chercher ses fichiers stage{1,2} −− vous pouvez utiliser la touche tabulation partout pour que Grub vous affiche les alternatives :

```
root (hd0,3)
```

Avertissement
La commande suivante écrasera votre chargeur de démarrage actuel. Ne lancez pas cette commande si ce n'est pas ce que vous voulez. Par exemple, vous pourriez utiliser un gestionnaire de démarrage autre pour gérer votre MBR (Master Boot Record). Dans ce cas, il serait probablement plus sensé d'installer Grub dans le secteur de boot de la partition LFS, auquel cas la commande deviendrait : **setup (hd0,3)**.

Ensuite, indiquez−lui de s'installer dans le MBR de hda :

```
setup (hd0)
```

Si tout va bien, Grub indiquera avoir trouvé ses fichiers dans /boot/grub. C'est tout ce qu'il y a à faire :

```
quit
```

Maintenant, nous avons besoin de créer un fichier "liste de menus" définissant le menu de démarrage de Grub :

```
cat > /boot/grub/menu.lst << "EOF"
# Debut de /boot/grub/menu.lst

# Par defaut, lance la premiere entree du menu.
default 0

# Attends 30 secondes avant de lancer le defaut.
timeout 30

# Utilise de jolis couleurs.
color green/black light-green/black

# La premiere entree concerne LFS.
title LFS 5.0
root (hd0,3)
kernel /boot/lfskernel root=/dev/hda4 ro
EOF
```

Vous pouvez souhaiter ajouter une entrée pour votre distribution hôte. Cela pourrait ressembler à ceci :

```
cat >> /boot/grub/menu.lst << "EOF"
title Red Hat
root (hd0,2)
kernel /boot/kernel-2.4.20 root=/dev/hda3 ro
initrd /boot/initrd-2.4.20
EOF
```

De même, s'il vous arrive de lancer Windows, l'entrée suivante devrait le permettre :

```
cat >> /boot/grub/menu.lst << "EOF"
title Windows
rootnoverify (hd0,0)
chainloader +1
EOF
```

Si **`info grub`** ne vous dit pas tout ce que vous souhaitez savoir, vous pouvez trouver plus d'informations concernant Grub sur son site web, situé sur http://www.gnu.org/software/grub.

Chapitre 9. La fin

La fin

Bien joué! Vous avez terminé d'installer votre système LFS. Cela vous a peut−être pris du temps, mais c'est une bonne chose que de l'avoir fait. Nous vous souhaitons de bien vous amuser avec votre nouveau système Linux rutilant.

Maintenant c'est le bon moment pour nettoyer les binaires de tous les symboles de débogage sur votre système LFS. Si vous n'êtes pas un programmeur et ne prévoyez pas de déboguer vos logiciels, alors vous serez certainement content de savoir que l'on peut gagner quelques dizaines de mégas en enlevant les symboles de débogage. Ce processus n'a pas d'autre inconvénient que de vous empêcher de déboguer votre logiciel à l'avenir, ce qui n'a pas d'importance si vous ne saviez pas comment le faire.

Précision : 98% des personnes qui utilisent la commande ci−dessous n'ont jamais eu de problème. Mais faites une sauvegarde de votre système LFS avant de lancer cette commande. Il y a une chance infime que cela se retourne contre vous et rende votre système inutilisable (essentiellement en détruisant les modules du noyau et les bibliothèques dynamiques partagées). Cela est plus souvent dû à des erreurs de frappes qu'à des problèmes avec les commandes utilisées.

Cela étant dit, l'option −−strip−debug que nous utilisons est plutôt sans dommage dans des circonstances normales. Cela ne débarrasse pas les fichiers de quoi que ce soit de vital. Il n'est sûr d'utiliser −−strip−all que sur des programmes classiques (ne pas utiliser sur des bibliothèques − elles seraient détruites) mais pas aussi sûr et l'espace gagné n'est pas plus grand. Mais si vous êtes limité en espace disque, quelques octets de plus peuvent aider, alors décidez vous−même. Référez vous, s'il vous plait, à la page man pour d'autres options de strip que vous pourriez utiliser. L'idée générale est de ne pas lancer strip sur des bibliothèques (autre que −−strip−debug) pour rester du côté sûr.

Si vous pensez vous lancer dans cette opération, une attention particulière est nécessaire pour s'assurer qu'aucun des binaires que vous allez modifier n'est en cours d'exécution, ceci incluant le shell bash actif. Donc, vous devez quitter l'environnement chroot et y rentrer de nouveau avec une commande chroot modifiée :

```
logout
chroot $LFS /tools/bin/env -i \
    HOME=/root TERM=$TERM PS1='\u:\w\$ ' \
    PATH=/bin:/usr/bin:/sbin:/usr/sbin \
    /tools/bin/bash --login
```

Maintenant, lancez la commande suivante :

```
/tools/bin/find /{,usr/,usr/local/}{bin,sbin,lib} -type f \
   -exec /tools/bin/strip --strip-debug '{}' ';'
```

Un grand nombre de fichiers seront indiqués comme non supportés à cause de leur format de fichier. La plupart d'entre eux sont des scripts au lieu d'être des binaires. Ces messages peuvent être ignorés sans problème.

Il pourrait être une bonne idée de créer un fichier `/etc/lfs−release`. Avec ce fichier, il vous est très facile (ainsi que pour nous si vous nous demandez de l'aide) de savoir quelle version de LFS vous avez

installé sur votre système. Créez ce fichier en lançant :

```
echo 5.0 > /etc/lfs-release
```

Enregistrez-vous

Vous voulez être enregistré comme utilisateur de LFS maintenant que vous avez terminé le livre ? Allez directement sur http://linuxfromscratch.org/cgi-bin/lfscounter.cgi et enregistrez-vous comme utilisateur LFS en entrant votre nom et la première version de LFS que vous ayez utilisée.

Replongeons-nous maintenant dans LFS...

Redémarrer le système

Maintenant que tous les logiciels ont été installés, il est temps de sortir de l'environnement chroot et de redémarrer l'ordinateur. Avant cela, démontons tous les systèmes de fichiers virtuels en lançant :

```
umount /proc
umount /dev/pts
```

Sortez de l'environnement chroot :

```
logout
```

De plus, maintenant que tous les logiciels ont été installé, il n'est plus besoin du répertoire `/tools`. Vous pouvez le supprimer. Comme cela supprimera aussi les copies temporaires de Tcl, Expect et DejaGnu, qui ont été utilisé pour lancer les tests de l'ensemble d'outils, vous aurez besoin de les recompiler et de les réinstaller sur votre système LFS si vous souhaitez utiliser ces programmes plus tard.

De même, vous pouvez maintenant souhaiter déplacer le contenu de `/sources` vers `/usr/src/packages` ou quelque chose de similaire (ou de les supprimer si vous les avez gravés sur un CD) et supprimer le répertoire.

Avant de redémarrer, démontons la partition LFS :

```
umount $LFS
```

Si vous aviez précédemment décidé de créer plusieurs partitions, vous aurez besoin de démonter les autres partitions avant de démonter $LFS, comme ceci :

```
umount $LFS/usr
umount $LFS/home
umount $LFS
```

Et maintenant, vous pouvez redémarrer votre système en lançant quelque chose comme :

```
/sbin/shutdown -r now
```

En supposant que le chargeur de démarrage Grub a été configuré comme indiqué précédemment, le menu par défaut devrait démarrer sur *LFS 5.0* automatiquement.

Après avoir redémarré, votre système LFS est prêt à l'emploi et vous pouvez commencer à ajouter vos propres logiciels.

Et maintenant?

Nous vous remercions d'avoir lu le livre LFS et espérons que vous l'avez trouvé utile.

Maintenant que vous avez terminé d'installer votre système LFS, vous êtes peut−être en train de vous demander "Et maintenant ?". Pour répondre à cette question, nous vous avons préparé une liste de ressources.

- Beyond Linux From Scratch

  Le livre Beyond Linux From Scratch couvre les procédures d'installation d'un grand nombre de logiciels en dehors du livre LFS. Le projet BLFS est disponible sur http://www.linuxfromscratch.org/blfs/.
- Astuces LFS (LFS Hints)

  Les actuces LFS sont une collection de petits documents éducatifs soumis par des volontaires à la communauté LFS. Ces astuces sont disponibles sur http://www.linuxfromscratch.org/hints/list.html.
- Listes de diffusion

  Il existe plusieurs listes de diffusion auxquelles vous pouvez vous abonner si vous cherchez de l'aide. Voir la section intitulée *Listes de diffusion* dans Chapitre 1 pour plus d'informations.
- Le Projet de Documentation Linux (Linux Documentation Project)

  Le but du LDP est de collaborer à tous les problèmes relatifs à la documentation sur Linux. Le LDP offre une large collection de HOWTO, guides et pages man; il est disponible sur http://www.tldp.org/.

IV. Partie IV − Annexes

Table des matières

Annexe A. Descriptions des packages et dépendances

Introduction

Dans cette annexe, les aspects suivants des packages installés dans ce livre sont décrits:

- le site officiel de téléchargement pour le package,
- ce que le package contient,
- ce que chaque programme du package réalise,
- ce que chaque package a besoin pour compiler.

Beaucoup d'informations sur ces packages (principalement les descriptions) proviennent des pages man, contenues dans ces packages. Nous n'incluons pas la page man entière, mais simplement les éléments principaux pour permettre la compréhension de l'utilité du package. Pour disposer de tous les détails sur un programme, rérérez−vous à sa page man ou à sa page info.

Certains packages sont documentés plus en profondeur que d'autres, parce que nous en savons plus sur ceux−là. Si quelque chose doit être ajoutés aux descriptions, n'hésitez surtout pas à nous envoyer un email à la liste de diffusion. Nous avons l'intention que cette liste contienne une description en profondeur de chaque package installé, mais nous ne pouvons pas le faire sans aide.

Remarquez aussi qu'actuellement, seul ce que fait un package est décrit. Pourquoi il doit être installé n'est pas encore couvert. Ceci pourra être ajouté plus tard.

Toutes les dépendances de chaque package installé avec ce livre sont décrites. Seront inclus les programmes de chaque package nécessaires pour compiler le package à installer.

Ce ne sont pas des dépendances lors de l'exécution, c'est−à−dire ceux dont vous avez besoin pour les lancer. Uniquement ceux nécessaire pour la compilation.

La liste de dépendances peut ne pas être à jour de temps en temps à cause de la version courant du package. Vérifier les dépendances demande un travail important, donc il peut y avaoir du temps avec une mise à jour. Mais souvent avec des mises à jour mineures d'un package, les dépendances d'installation changent peu, donc elles seront correctes dans la plupart des cas. Quand nous mettons à jour vers une nouvelle version majeure, nous nous assurons que les dépendances ont aussi été vérifié.

Autoconf

Pour les instructions d'installation, voir la section intitulée *Installer Autoconf−2.57* dans Chapitre 6.

Site officiel de téléchargement

Autoconf (2.57) : ftp://ftp.gnu.org/gnu/autoconf/

Contenu d'Autoconf

Autoconf crée des scripts shell qui configurent automatiquement du code source.

Programmes installés : autoconf, autoheader, autom4te, autoreconf, autoscan, autoupdate et ifnames

Descriptions courtes

autoconf est un outil destiné à produire des scripts shell qui configurent automatiquement les packages contenant les sources et ceci afin de les adapter à de nombreux systèmes de type Unix. Les scripts de configuration qu'il produit sont indépendants −− les lancer ne nécessite pas le programme autoconf.

autoheader est un outil de création de fichiers modèle des déclarations C #define destinés à être utilisé par l'utilitaire configure.

autom4te est un emballage du processeur de macros M4.

autoscan peut aider à créer un fichier `configure.in` pour un package logiciel. Il examine les fichiers sources du répertoire, recherchant les problèmes de portabilité habituels et crée un fichier `configure.scan` utilisé comme préliminaire au `configure.in` pour ce package.

autoupdate modifie le fichier `configure.in`, qui appelle toujours les macros autoconf par leurs anciens noms, pour utiliser les noms actuels des macros.

ifnames peut être utile lors de l'écriture d'un fichier `configure.in` pour un package logiciel. Il écrit les identifiants que le package utilise dans des directives conditionnelles du pré−processeur C. Si un package a déjà été configuré pour avoir une certaine portabilité, ce programme peut aider à identifier ce que son script **`configure`** doit vérifier. Ceci peut aider à combler certains manques dans un fichier `configure.in` généré par autoscan.

Dépendances d'installation d'Autoconf

Autoconf dépend de Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

Automake

Pour les instructions d'installation, voir la section intitulée *Installer Automake−1.7.6* dans Chapitre 6.

Site Officiel de Téléchargement

Automake (1.7.6) : ftp://ftp.gnu.org/gnu/automake/

Contenu d'Automake

Automake génère des fichiers Makefile.in, à utiliser avec Autoconf.

Programmes installés : acinstall, aclocal, aclocal−1.6, automake, automake−1.6, compile, config.guess, config.sub, depcomp, elisp−comp, install−sh, mdate−sh, missing, mkinstalldirs, py−compile, ylwrap

Descriptions courtes

acinstall est un script qui installe les fichiers M4 de type aclocal.

aclocal génère des fichiers `aclocal.m4` basés sur le contenu des fichiers `configure.in`.

automake est un outil pour générer automatiquement des `Makefile.in` à partir des fichiers nommés `Makefile.am`. Pour créer tous les fichiers `Makefile.in` d'un, lancez ce programme dans le répertoire de haut niveau. En parcourant `configure.in`, il trouve automatiquement chaque `Makefile.am` approprié pour générer le `Makefile.in` correspondant.

compile est un emballage pour les compilateurs.

config.guess est un script essayant de deviner le triplé canonique pour la construction, l'hôte ou l'éarchitecture de la cible.

config.sub est un script contenant une sous−routine de validation de configuration.

depcomp est un script pour compiler un programme de façon à ce que la sortie souhaitée soit générée, mais aussi d'avoir les informations de dépendances

elisp−comp compile ("byte−code") le code Lisp Emacs.

install−sh est un script qui installe un programme, un script ou un fichier de données.

mdate−sh est un script qui affiche l'heure de modification d'un fichier ou d'un répertoire.

missing est un script agissant comme une interface commune pour les programmes GNU manquants lors d'une installation.

mkinstalldirs est un script créant une hiérarchie de répertoires.

py−compile compile un programme Python.

ylwrap est comme un emballage pour lex et yacc.

Dépendances d'installation d'Automake

Automake dépend de Autoconf, Bash, Coreutils, Diffutils, Grep, M4, Make, Perl, Sed.

Bash

Pour les instructions d'installation, voir la section intitulée *Installer Bash−2.05b* dans Chapitre 6.

Site Officiel de téléchargement

Bash (2.05b): ftp://ftp.gnu.org/gnu/bash/ Correctif Bash (2.05b): http://www.linuxfromscratch.org/patches/lfs/5.0/bash−2.05b−2.patch

Contenu de Bash

Bash, acronyme de Bourne−Again SHell, est un interpréteur de commande très répandu sur les systèmes Unix. Il se charge d'interpréter et d'évaluer les commandes que l'utilisateur entre sur l'entrée standard (le clavier), comme par exemple lancer un programme.

Programmes installés : bash, sh (lien vers bash) et bashbug

Descriptions courtes

bash est un interpréteur de commande très répandu. Il réalise tout sortes d'expansions et de substitutions sur une ligne de commande donnée avant de l'exécuter, rendant de ce fait cet interpréteur un outil très puissant.

bashbug est un script shell aidant l'utilisateur à composer et à envoyer un rapport de bogue concernant bash en un format standard.

sh est un lien symbolique vers le programme bash. Quand bash est invoqué en tant que sh, il essaie de simuler le comportement des versions historiques de sh, tout en restant conforme au standard POSIX.

Dépendances d'installation de Bash

Bash dépend de Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Ncurses, Sed.

Binutils

Pour les instructions d'installation, voir la section intitulée *Installer Binutils−2.14* dans Chapitre 6.

Site Officiel de Téléchargement

Binutils (2.14) : ftp://ftp.gnu.org/gnu/binutils/

Contenu de Binutils

Binutils est une collection d'outils de développement de logiciels contenant un éditeur de liens, un assembleur et d'autres outils pour travailler avec des fichiers objet et des archives.

Programmes installés : addr2line, ar, as, c++filt, gprof, ld, nm, objcopy, objdump, ranlib, readelf, size, strings et strip

Bibliothèques installés : libiberty.a, libbfd.[a,so] et libopcodes.[a,so]

Descriptions courtes

addr2line traduit les adresses du programme en des noms de fichiers et des numéros de ligne. Avec une adresse et le nom d'un exécutable, il utilise les informations de déboguage dans l'exécutable pour savoir quel fichier source et quel numéro de ligne sont associés à cette adresse.

ar crée, modifie et extrait à partir des archives. Une archive est un simple fichier contenant une collection d'autres fichiers dans une structure qui rend possible la recherche des fichiers individuels originaux (appelés membres de l'archive).

as est un assembleur. Il assemble la sortie de gcc en des fichiers objets.

c++filt est utilisé par l'éditeur de liens pour filtrer les symbôles C++ et Java, en empêchant ainsi les fonctions surchargés de se mélanger.

gprof affiche les données d'appels profilés.

ld est un éditeur de liens. Il combine un certain nombre de fichiers objets et archives, déplaçant leur données et rangeant les symbôles de référence.

nm liste les symbôles trouvés dans un fichier objet donné.

objcopy est utilisé pour traduire un type de fichier objet en un autre.

objdump affiche des informations sur le fichier objet donné, avec des options contrôlant les informations particulières à afficher. L'information affichée est utile pratiquement seulement aux programmeurs qui travaillent sur des outils de compilation.

ranlib génère un index du contenu d'une archive et le stocke dans l'archive. L'index liste tous les symbôles définis par les membres de l'archive qui sont des fichiers object relocalisables.

readelf affiche des informations sur les binaires de type elf.

size liste les tailles des sections et le total pour les fichiers objets donnés.

strings affiche pour chaque fichier indiqué les séquences de caractères imprimables qui sont au moins de la longueur spécifiée (par défaut à 4). Pour les fichiers objets, il affiche par défaut seulement les chaînes des sections d'initialisation et de chargement. Pour les autres types de fichiers, il parcourt le fichier entier.

strip supprime les symbôles des fichiers objets.

libiberty contient des routines utilisées par différents programmes GNU dont getopt, obstack, strerror, strtol et strtoul.

libbfd est la bibliothèque du descripteur de fichier binaire (Binary File Descriptor).

libopcodes est une bibliothèque pour traiter les opcodes. Elle est utilisée pour construire des utilitaires comme objdump. Les opcodes sont des versions "textes lisibles" des instructions pour le processeur.

Dépendances d'installation de Binutils

Binutils dépend de Bash, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Bison

Pour les instructions d'installation, voir la section intitulée *Installer Binutils−2.14* dans Chapitre 6.

Site officiel de téléchargement

Bison (1.875) : ftp://ftp.gnu.org/gnu/bison/ Correctif "Bison Attribute" : http://www.linuxfromscratch.org/patches/lfs/5.0/bison−1.875−attribute.patch

Contenu de Bison

Bison est un générateur de l'analyseur, un remplacement pour yacc. Bison génère un programme analysant la structure d'un fichier texte.

Programmes installés : bison et yacc

Bibliothèques installés : liby.a

Descriptions courtes

bison génère, à partie d'une série de règles, un programme pour analyser la structure de fichiers texte. Bison est un remplacement pour yacc (Yet Another Compiler Compiler).

yacc est un emballage pour bison, nécessaire à certains programmes qui appelent toujours yacc au lieu de bison. Il appelle bison avec l'option −y.

liby.a est la bibliothèque Yacc contenant l'implémentation des fonctions yyerror et main compatibles avec Yacc. Cette bibliothèque n'est habituellement pas très utile mais POSIX la réclame.

Dépendances d'installation de Bison

Bison dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make, Sed.

Bzip2

Pour les instructions d'installation, voir la section intitulée *Installer Bzip2−1.0.2* dans Chapitre 6.

Site officiel de téléchargement

Bzip2 (1.0.2): http://sources.redhat.com/bzip2/

Contenu de Bzip2

Bzip2 est un compresseur de fichiers qui arrive habituellement à une meilleure compression que ne le fait **`gzip`** traditionel.

Programmes installés : bunzip2 (lien vers bzip2), bzcat (lien vers bzip2), bzcmp, bzdiff, bzegrep, bzfgrep, bzgrep, bzip2, bzip2recover, bzless et bzmore

Bibliothèques installées : libbz2.a, libbz2.so (lien vers libbz2.so.1.0), libbz2.so.1.0 (lien vers libbz2.so.1.0.2) et libbz2.so.1.0.2

Descriptions courtes

bunzip2 décompresse des fichiers bzip.

bzcat décompresse vers la sortie standard.

bzcmp lance cmp sur des fichiers bzip.

bzdiff lance diff sur des fichiers bzip.

bzgrep et ses amis lance grep sur des fichiers bzip.

bzip2 compresse des fichiers en utilisant l'algorithme de compression par tri de blocs de Burrows−Wheeler avec l'encodage de Huffman. Le taux de compression est généralement bien meilleur que celui obtenu par des compresseurs plus conventionnels utilisant LZ77/LZ78, comme **`gzip`**.

bzip2recover essaie de récupérer des données à partir de fichiers bzip2 endommagés.

bzless lance less sur des fichiers bzip.

bzmore lance more sur des fichiers bzip.

libbz2* est la bibliothèque implémentant lossless, une compression de données par tri de blocs utilisant l'algorithme de Burrows−Wheeler.

Dépendances d'installation de Bzip2

Bzip2 dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Make.

Coreutils

Pour les instructions d'installation, voir la section intitulée *Installer Coreutils−5.0* dans Chapitre 6.

Emplacement officiel de téléchargement

Coreutils (5.0): ftp://ftp.gnu.org/gnu/coreutils/ Correctif "Coreutils Hostname" : http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−hostname−2.patch
Correctif "Coreutils Uname" : http://www.linuxfromscratch.org/patches/lfs/5.0/coreutils−5.0−uname.patch

Contenu de Coreutils

Le package Coreutils contient un grand ensemble d'utilitaires shell basiques.

Programmes installés : basename, cat, chgrp, chmod, chown, chroot, cksum, comm, cp, csplit, cut, date, dd, df, dir, dircolors, dirname, du, echo, env, expand, expr, factor, false, fmt, fold, groups, head, hostid, hostname, id, install, join, kill, link, ln, logname, ls, md5sum, mkdir, mkfifo, mknod, mv, nice, nl, nohup, od, paste, pathchk, pinky, pr, printenv, printf, ptx, pwd, readlink, rm, rmdir, seq, sha1sum, shred, sleep, sort, split, stat, stty, su, sum, sync, tac, tail, tee, test, touch, tr, true, tsort, tty, uname, unexpand, uniq, unlink, uptime, users, vdir, wc, who, whoami et yes

Descriptions courtes

basename supprime tout chemin et un suffixe indiqué au nom de fichier donné en argument.

cat concatène des fichiers vers la sortie standard.

chgrp change le groupe de chaque fichier indiqué par le groupe donné en argument. Le groupe peut être donné soit par son nom soit par son identifiant numérique.

chmod change les droits de chaque fichier indiqué par le mode donné en argument. Ce mode peut être soit une représentation symbôlique des modifications à effectuer soit un nombre octal représentant les nouveaux droits.

chown change l'utilisateur et/ou le groupe de chaque fichier indiqué sur la ligne de commande avec la paire utilisateur:groupe.

chroot lance une commande donné avec un répertoire indiqué comme étant le nouveau répertoire `/`. La commande peut être un shell interactif. Sur la plupart des systèmes, seul *root* peut faire cela.

cksum affiche la somme de contrôle CRC et le nombre d'octets de chaque fichier spécifié.

comm compare deux fichiers triés, affichant sur trois colonnes les lignes uniques et les lignes communes.

cp copie des fichiers.

csplit divise un fichier donné en plusieurs nouveaux petits fichiers, les séparant suivant des modèles ou des numéros de ligne donnés et affichant le nombre d'octets de chaque fichier.

cut affiche des parties de lignes, en sélectionnant les parties suivant des champs ou des positions.

date affiche l'heure actuelle dans le format demandé, ou initialise la date système.

dd copie un fichier en utilisant la taille de bloc indiquée et le nombre tout en réalisant optionnellement des conversions.

df rapporte l'espace disque disponible (et utilisé) pour tous les systèmes de fichiers montés, ou seulement des systèmes de fichiers contenant les fichiers indiqués en ligne de commande.

dir est identique à ls.

dircolors affiche les commandes pour initialiser la variable d'environnement LS_COLOR, pour changer le schéma des couleurs utilisé par ls.

dirname conserve uniquement les noms des répertoires d'un fichier donné.

du rapporte l'espace disque utilisé par le répertoire actuel ou par chaque répertoire demandé en incluant leurs sous−répertoires ou par chaque fichier indiqué.

echo affiche les chaînes indiquées.

env lance une commande dans un environnement modifié.

expand convertit les tabulations en espaces.

expr evalue des expressions.

factor affiche les nombres premiers de nombres entiers indiqués sur la ligne de commande.

false ne fait rien à part renvoyer un code de sortie indiquant un échec.

fmt reformatte les paragraphes dans les fichiers donnés.

fold coupe les lignes des fichiers indiqués.

groups rapporte les groupes auxquels appartient un utilisateur.

head affiche les dix premières lignes (ou le nombre de lignes indiquées) de chaque fichier sur la ligne de commande.

hostid affiche l'identifiant numérique (hexadécimal) de l'hôte.

hostname affiche ou initialise le nom de l'hôte.

id rapporte l'identifiant effectif de l'utilisateur, de ses groupes.

install copie les fichiers tout en configurant les droits et, si possible, leur propriétaire utilisateur et groupe.

join fusionner les lignes de deux fichiers ayant un champ commun.

kill termine un processus donné.

link crée un lien physique avec le nom du fichier donné.

ln crée un lien physique ou symbolique entre des fichiers.

logname rapporte le nom de connexion de l'utilisateur actuel.

ls liste le contenu de chaque répertoire demandé. Par défaut, il ordonne les fichiers et sous−répertoires alphabêtiquement.

md5sum affiche ou vérifie la somme de contrôle MD5.

mkdir crée des répertoires avec les noms donnés.

mkfifo crée des FIFOs avec les noms donnés.

mknod crée des noeuds périphériques avec les noms donnés. Un noeud périphérique est un fichier spécial mode caractère ou mode bloc ou encore un FIFO.

mv déplace ou renomme des fichiers ou des répertoires.

nice lance un programme en modifiant sa priorité.

nl compte les lignes des fichiers donnés.

nohup lance une commande immunisée contre les hangups, dont la sortie est redirigé vers un fichier de traces.

od affiche les fichiers dans un format octal ou autre.

paste joint deux fichiers donnés, assemblant séquentiellement les lignes correspondantes face à face, en les séparant avec des tabulations.

pathchk vérifie que les noms de fichier sont valides ou portables.

pinky est un client finger léger. Il affiche quelques informations sur les utilisateurs indiqués.

pr gère la pagination et le colonnage des fichiers en vue d'une impression.

printenv affiche l'environnement.

printf affiche les arguments spécifiés suivant le format défini −− tout à fait comme la fonction C printf.

ptx produit un index permutté à partir du contenu des fichiers indiqués avec chaque mot clé dans son contexte.

pwd indique le nom du répertoire actuel.

readlink indique la valeur du lien symbolique précisé sur la ligne de commande.

rm supprime des fichiers ou répertoires.

rmdir supprime des répertoires, à condition qu'ils soient vides.

seq affiche une séquence de nombres compris dans une certaine échelle avec une incrémentation spécifiée.

sha1sum affiche ou vérifie les sommes de contrôle 160 bits SHA1.

shred écrase les fichiers indiqués en y enregistrant de façon répété des modèles étranges pour rendre la récupération des données particulièrement difficile.

sleep stoppe le programme pour un certain temps.

sort trie les lignes des fichiers donnés en argument.

split divise le fichier donné en plusieurs parties par une certaine taille ou par un nombre de lignes.

stty initialise ou affiche les paramètrages de la ligne série du terminal.

su lance un shell en substituant l'identifiant de l'utilisateur et du groupe.

sum affiche la somme de contrôle et le nombre de blocs pour chaque fichier indiqué sur la ligne de commande.

sync vide les tampons du système de fichiers. Il force les blocs modifiés à être enregistrés sur le disque et met à jour le superbloc.

tac concatène les fichiers en arguments en les inversant.

tail affiche les dix dernières lignes (ou le nombre de lignes indiqué) pour chaque fichier spécifié sur la ligne de commande.

tee lit à partir de l'entrée standard et écrit à la fois sur la sortie standard et dans les fichiers spécifiés.

test compare des valeurs et vérifie les types des fichiers.

touch modifie les heures des fichiers, en initialisant les heures et dates d'accès et de modification des fichiers indiqués sur la ligne de commande à l'heure et la date actuelle. Les fichiers n'existant pas sont crées avec une taille nulle.

tr transforme, modifie et supprime les caractères indiqués à partir de l'entrée standard.

true ne fait rien à part renvoyer une valeur de retour indiquant le succès.

tsort réalise un tri topologique. Il ecrit une liste totalement ordonnée suivant l'ordre partiel d'un fichier donné.

tty affiche le nom du fichier du terminal connecté à l'entrée standard.

uname affiche des informations sur le système.

unexpand convertit des espaces en tabulations.

uniq conserve tous les lignes successives identiques.

unlink supprime le fichier spécifié.

uptime affiche le temps où la machine est resté allumée, le nombre d'utilisateurs connecté et la charge système.

users affiche le nom des utilisateurs actuellement connectés.

vdir est identique à ls −l.

wc affiche le nombre de lignes, mots et octets pour chaque fichier donné, ainsi que le nombre total de lignes lorsque plus d'un fichier est indiqué.

who affiche qui est connecté.

whoami affiche le nom de l'utilisateur associé à l'identifiant utilisateur actuel.

yes affiche 'y' ou une chaîne spécifiée de façon répété, jusqu'à ce que cette tâche soit tuée.

Dépendances d'installation de Coreutils

Coreutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed.

DejaGnu

Pour les instructions d'installation, voir la section intitulée *Installer DejaGnu−1.4.3* dans Chapitre 5.

Site officiel de téléchargement

DejaGnu (1.4.3): ftp://ftp.gnu.org/gnu/dejagnu/

Contenu de DejaGnu

Le package DejaGnu contient un groupe de travail pour tester d'autres programmes.

Programme installé : runtest

Descriptions courtes

(Dernière vérification effectuée auprès de la version 1.4.3.)

runtest est un script d'emballage qui trouve le bon shell expect puis appelle DejaGnu.

Dépendances d'installation de DejaGnu

Dejagnu dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Diffutils

Pour les instructions d'installation, voir la section intitulée *Installer Diffutils−2.8.1* dans Chapitre 6.

Site officiel de

Diffutils (2.8.1) : ftp://ftp.gnu.org/gnu/diffutils/

Contenu de Diffutils

Les programmes de ce package vous montre les différences entre les deux fichiers ou répertoires. Son utilisation la plus commune est la création de correctifs.

Programmes installés : cmp, diff, diff3 et sdiff

Descriptions courtes

cmp compare deux fichiers et rapporte si et où ils différent.

diff compare deux fichiers ou répertoires et rapporte les lignes différentes.

diff3 compare trois fichiers ligne par ligne.

sdiff assemble deux fichiers et affiche interactivement les résultats.

Dépendances d'installation de Diffutils

Diffutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

E2fsprogs

Pour les instructions d'installation, voir la section intitulée *Installer E2fsprogs−1.34* dans Chapitre 6.

Site officiel de téléchargement

E2fsprogs (1.34) : ftp://download.sourceforge.net/pub/sourceforge/e2fsprogs/
http://download.sourceforge.net/e2fsprogs/

Contenu de E2fsprogs

E2fsprogs apporte des utilitaires du système de fichiers ext2. Il supporte aussi le système de fichiers ext3 avec le support de la journalisation.

Programmes installés : badblocks, blkid, chattr, compile_et, debugfs, dumpe2fs, e2fsck, e2image, e2label, findfs, fsck, fsck.ext2, fsck.ext3, logsave, lsattr, mk_cmds, mke2fs, mkfs.ext2, mkfs.ext3, mklost+found, resize2fs, tune2fs et uuidgen.

Bibliothèques installées : libblkid.[a,so], libcom_err.[a,so], libe2p.[a,so], libext2fs.[a,so], libss.[a,so] et libuuid.[a,so]

Descriptions courtes

badblocks recherche un périphérique (habituellement une partition disque) pour des mauvais blocs.

blkid est un utilitaire en ligne de commande permettant de localiser et d'afficher les attributs des périphériques blocs.

chattr modifie les attributs de fichiers sur un système de fichiers ext2.

compile_et est un compilateur de table d'erreurs. Il convertit une table de noms de code d'erreurs et de messages en un fichier source C possible à utiliser avec la bibliothèque com_err.

debugfs est un débuggeur de systèmes de fichiers. Il est utilisable pour examiner et changer l'état d'un système de fichiers ext2.

dumpe2fs affiche le super bloc et les informations sur le groupe de blocs pour le système de fichier présent sur un périphérique donné.

e2fsck est utilisé pour vérifier, et optionnellement réparer, des systèmes de fichiers ext2 et ext3.

e2image est utilisé pour sauver des données critiques d'un système de fichiers ext2 vers un fichier.

e2label affichera ou modifiera le label de système de fichiers sur le système de fichiers ext2 présent sur le périphérique indiqué.

findfs trouve un système de fichiers par son label ou son UUID.

fsck est utilisé pour vérifier, et optionnellement réparer, les systèmes de fichiers. Par défaut, il vérifie les systèmes de fichiers listés dans `/etc/fstab`

logsave sauvegarde la sortie d'une commande dans un journal.

lsattr liste les attributs de fichier sur un système de fichiers ext2.

mk_cmds convertit une table de noms de commandes et de messages d'aide en un fichier source C utilisable avec la bibliothèque `libss`.

mke2fs est utilisé pour créer un système de fichiers ext2 sur un périphérique donné.

mklost+found est utilisé pour créer un répertoire `lost+found` sur un système de fichiers ext2. Il pré−alloue des blocs du disque pour ce répertoire ce qui allège le travail de e2fsck.

resize2fs peut être utilisé pour agrandir ou réduire un système de fichiers ext2.

tune2fs est utilisé pour ajuster les paramètres d'un système de fichiers ext2.

uuidgen crée de nouveaux identifieurs uniques (Universally Unique IDentifiers, ou UUID). Chaque nouvel UUID peut être raisonnablement considéré unique parmi tous les UUID créés sur le système local, ou sur un autre, dans le passé et dans le futur.

libblkid contient des routines pour une identification de périphérique et pour une extraction de modèles.

libcom_err est la routine d'affichage des erreurs communes.

libe2p est utilisé par dumpe2fs, chattr et lsattr.

libext2fs contient des routines pour activer des programmes utilisateurs manipulant des systèmes de fichiers ext2.

libss est utilisé par debugfs.

libuuid contient des routines pour générer des identifieurs uniques pour les objets pouvant rester accessibles en dehors du système local.

Dépendances d'installation d'E2fsprogs

E2fsprogs dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Glibc, Grep, Make, Sed, Texinfo.

Ed

Pour les instructions d'installation, voir la section intitulée *Installer Ed−0.2* dans Chapitre 6.

Site officiel de téléchargement

Ed (0.2) : ftp://ftp.gnu.org/gnu/ed/ Correctif "Ed Mkstemp" : http://www.linuxfromscratch.org/patches/lfs/5.0/ed−0.2−mkstemp.patch

Contenu d'Ed

GNU ed est un éditeur par lignes compatible POSIX.

Programmes installés : ed et red (lien vers ed)

Descriptions courtes

ed est un éditeur de texte par ligne. Il est utilisé pour créer, afficher, modifier et réaliser d'autres manipulations sur les fichiers texte.

red est un ed restreint : il ne peut qu'éditer les fichiers du répertoire courant et ne peut pas exécuter des commandes du shell.

Dépendances d'installation de Ed

Ed dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Expect

Pour les instructions d'installation, voir la section intitulée *Installer Expect−5.39.0* dans Chapitre 5.

Site officiel de téléchargement

Expect (5.39.0) : http://expect.nist.gov/src/ Correctif "Expect Spawn" : http://www.linuxfromscratch.org/patches/lfs/5.0/expect−5.39.0−spawn.patch

Contenu de Expect

Le package Expect contient un programme exécutant un dialogue programmé avec d'autres programmes interactifs.

Programme installé : expect

Bibliothèque installée : libexpect5.39.a

Descriptions courtes

expect "parle" aux autres programmes interactifs suivant un script.

Dépendances d'installation d'Expect

Expect dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, Tcl.

File

Pour les instructions d'installation, voir la section intitulée *Installer File−4.04* dans Chapitre 6.

Site officiel de téléchargement

File (4.04) : ftp://ftp.gw.com/mirrors/pub/unix/file/ Autre site de téléchargement : ftp://gaosu.rave.org/pub/linux/lfs/

Contenu de File

File est un utilitaire servant à déterminer les types de fichiers.

Programme installé : file

Bibliothèque installée : libmagic.[a,so]

Descriptions courtes

file essaie de classifier chaque fichier donné. Il fait cela en exécutant plusieurs tests: tests sur le système de fichiers, tests sur des nombres magiques et tests de langages? Le premier test réussissant détermine le résultat.

libmagic contient des routines pour la reconnaissance de nombres magiques utilisées par le programme file.

Dépendances d'installation de File

File dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed, Zlib.

Findutils

Pour les instructions d'installation, voir la section intitulée *Installer Findutils−4.1.20* dans Chapitre 6.

Site officiel de téléchargement

Findutils (4.1.20): ftp://alpha.gnu.org/gnu/findutils/

Contenu de Findutils

Le package Findutils contient les programmes de recherche de fichiers, soit en direct (en faisant une recherche récursive en direct à travers les répertoires et seulement en affichant des fichiers qui remplissent ses spécifications) soit en cherchant à travers une base de données.

Programmes installés : bigram, code, find, frcode, locate, updatedb et xargs

Descriptions courtes

bigram était auparavant utilisé pour créer des bases de données locate.

code était auparavant utilisé pour créer des bases de données locate. Il est l'ancêtre de frcode.

find cherche des fichiers dans des répertoires donnés en suivant des critère spécifiés.

frcode est appelé par updatedb pour compresser la liste des noms de fichiers. Il utilise la compression front, réduisant la taille de la base de données d'un facteur de 4 à 5.

locate cherche au travers de la base de données de noms de fichiers et affiche les noms contenant une chaîne donnée ou correspondant à un modèle spécifié.

updatedb met à jour la base de données locate. Il scanne le système de fichiers complet (en incluant les autres systèmes de fichiers déjà montés, sauf indication du contraire) et place chaque nom de fichier qu'il trouve dans la base de données.

xargs peut être utilisé pour exécuter une commande donnée à une liste de fichiers.

Dépendances d'installation de Findutils

Findutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Flex

Pour les instructions d'installation, voir la section intitulée *Installer Flex−2.5.4a* dans Chapitre 6.

Site officiel de téléchargement

Flex (2.5.4a): ftp://ftp.gnu.org/non−gnu/flex/

Contenu de Flex

Le package Flex est utilisé pour générer des programmes de reconnaissance de modèles dans du texte.

Programmes installés : flex, flex++ (lien vers flex) et lex

Bibliothèque installée : libfl.a

Descriptions courtes

flex est un outil générant des programmes de reconnaissance de modèles dans un texte. La reconnaissance de modèles est utile dans beaucoup d'applications. A partir d'un ensemble de règles sur ce qui est à rechercher, flex crée un programme recherchant ces modèles. La raison principale pour utiliser flex est que cela est bien plus facile de spécifier les règles que d'écrire directement le programme de recherche des modèles.

flex++ appelle une version de flex utilisée uniquement par des scanners C++.

libfl est la bibliothèque flex.

Dépendances d'installation de Flex

Flex dépend de Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, M4, Make, Sed.

Gawk

Pour les instructions d'installation, voir la section intitulée *Installer Gawk−3.1.3* dans Chapitre 6.

Site officiel de téléchargement

Gawk (3.1.3): ftp://ftp.gnu.org/pub/gnu/gawk/ Correctif "Gawk Libexecdir" : http://www.linuxfromscratch.org/patches/lfs/5.0/gawk−3.1.3−libexecdir.patch

Contenu de Gawk

Gawk est une implémentation de awk qui est utilisée pour manipuler des fichiers texte.

Programmes installés : awk (lien vers gawk), gawk, gawk−3.1.3, grcat, igawk, pgawk, pgawk−3.1.3 et pwcat.

Descriptions courtes

gawk est un programme de manipulation de fichiers texte. Il s'agit de l'implémentation GNU de awk.

grcat affiche le base de données des groupes `/etc/group`.

igawk donne à gawk la capacité d'inclure des fichiers.

pgawk est la version de profilage de gawk.

pwcat affiche la base de données des mots de passe `/etc/passwd`.

Dépendances d'installation de Gawk

Gawk dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

GCC

Pour les instructions d'installation, voir la section intitulée *Installer GCC−3.3.1* dans Chapitre 6.

Site officiel de téléchargement

GCC (3.3.1)" : ftp://ftp.gnu.org/pub/gnu/gcc/ Correctif "GCC No−Fixincludes" : http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−no_fixincludes−2.patch Correctif "GCC Specs" : http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−specs−2.patch
Correctif "GCC Suppress−Libiberty" : http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−3.3.1−suppress−libiberty.patch GCC−2 (2.95.3): ftp://ftp.gnu.org/pub/gnu/gcc/ GCC−2: http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−2.patch
Correctif "GCC−2 No−Fixincludes" : http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−no−fixinc.patch
Correctif "GCC−2 Return−Type" : http://www.linuxfromscratch.org/patches/lfs/5.0/gcc−2.95.3−returntype−fix.patch

Contenu de GCC

Le package GCC contient le compilateur GNU, incluant les compilateurs C et C++.

Programmes installés : c++, cc (lien vers gcc), cc1, cc1plus, collect2, cpp, g++, gcc, gccbug et gcov

Bibliothèques installées : libgcc.a, libgcc_eh.a, libgcc_s.so, libstdc++.[a,so] et libsupc++.a

Descriptions courtes

cpp est le préprocesseur C. Il est utilisé par le compilateur pour avoir les instructions #include et #define étendues dans les fichiers source.

g++ est le compilateur C++.

gcc est le compilateur C. Il est utilisé pour traduire le code source d'un programme en code assembleur.

gccbug est un script shell utilisé pour aider à la création de bons rapports de bug.

gcov est un outil de tests. Il est utilisé pour analyser des programmes dans l'espoir de trouver où des optimisations auraient le plus d'effet.

libgcc* contient le support en exécution de gcc.

libstdc++ est la bibliothèque standard C++. Elle contient des fonctions fréquemment utilisées.

libsupc++ apporte des routines de support pour le langage de programmation c++.

Dépendances d'installation de GCC

GCC dépend de Bash, Binutils, Coreutils, Diffutils, Findutils, Gawk, Gettext, Glibc, Grep, Make, Perl, Sed, Texinfo.

Gettext

Pour les instructions d'installation, voir la section intitulée *Installer Gettext−0.12.1* dans Chapitre 6.

Site officiel de téléchargement

Gettext (0.12.1) : ftp://ftp.gnu.org/gnu/gettext/

Contenu de Gettext

Le package Gettext est utilisé pour l'internationalisation et la localisation. Les programmes peuvent êtres compilés avec le support de la langue native ('Native Language Support' ou NLS) qui leur permettent d'afficher les messages dans la langue native de l'utilisateur.

Programmes installés : autopoint, config.charset, config.rpath, gettext, gettextize, hostname, msgattrib, msgcat, msgcmp, msgcomm, msgconv, msgen, msgexec, msgfilter, msgfmt, msggrep, msginit, msgmerge, msgunfmt, msguniq, ngettext, project−id, team−address, trigger, urlget, user−email et xgettext

Bibliothèques installées : libasprintf[a,so], libgettextlib[a,so], libgettextpo[a,so] et libgettextsrc[a,so]

Descriptions courtes

autopoint copie les fichiers de l'infrastructure standard de gettext dans un package source.

config.charset affiche une table des alias de codage des caractères dépendante du système.

config.rpath affiche un ensemble, dépendant du système, des variables, décrivant comment initialiser le chemin de recherche des bibliothèques partagées à l'exécution dans un exécutable.

gettext traduit un message en langue naturel dans la langue de l'utilisateur en recherchant la traduction dans un catalogue de messages.

gettextize copie tous les fichiers standards Gettext dans le répertoire principal d'un package, pour commencer sa traduction.

hostname affiche le nom d'hôte réseau de plusieurs façons.

msgattrib filtre les messages du catalogue de traduction suivant leurs attributs et manipule ces derniers.

msgcat concatène et assemble les fichiers `.po` donnés.

msgcmp compare deux fichiers `.po` pour vérifier que les deux contiennent le même ensemble de chaînes msgid.

msgcomm trouve les messages communs aux fichiers `.po` indiqués.

msgconv convertit un catalogue de traduction en un codage de caractères différent.

msgen crée un catalogue de traduction anglais.

msgexec applique une commande pour toutes les traductions sur un catalogue de traduction.

msgfilter applique un filtre sur toutes les traductions d'un catalogue de traduction.

msgfmt génère un catalogue de messages binaires à partir d'un catalogue de traduction.

msggrep extrait tous les messages d'un catalogue de traduction correspondant à un modèle donné ou appartenant à des quelques fichiers sources donnés.

msginit crée un nouveau fichier `.po` en initialisant les méta−informations avec des valeurs provenant de l'environnement utilisateur.

msgmerge combine deux traductions brutes en un seul fichier.

msgunfmt décompile un catalogue de messages binaires en un texte brut de traduction.

msguniq unifie les traductions dupliquées dans un catalogue de traduction.

ngettext affiche les traductions en langage natif d'un message texte dont la forme graphique dépend d'un nombre.

xgettext extrait les lignes de messages traduisibles à partir des fichiers sources donnés, pour réaliser le premier modèle de traduction.

libasprintf defines the autosprintf class which makes C formatted output routines usable in C++ programs, for use with the <string> strings and the <iostream> streams.

libgettextlib est une bibliothèque privée contenant des routines communes utilisées par les différents programmes gettext. Elle n'est pas prévue pour un usage général.

libgettextpo est utilisé pour écrire des programmes spécialisés dans l'utilisation des fichiers PO. Cette bibliothèque est utilisée lorsque les applications standard fournies avec gettext, comme msgcomm, msgcmp, msgattrib et msgen, ne suffisent pas.

libgettextsrc est une bibliothèque privée contenant des routines communes utilisées par les différents programmes gettext. Ils n'ont pas d'utilité dans un cadre général.

Dépendances d'installation de Gettext

Gettext dépend de Bash, Binutils, Bison, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Glibc

Pour les instructions d'installation, voir la section intitulée *Installer Glibc−2.3.2* dans Chapitre 6.

Site officiel de téléchargement

Glibc (2.3.2) : ftp://ftp.gnu.org/gnu/glibc/ Glibc−linuxthreads (2.3.2) : ftp://ftp.gnu.org/gnu/glibc/
Correctif "Glibc Sscanf" : http://www.linuxfromscratch.org/patches/lfs/5.0/glibc−2.3.2−sscanf−1.patch

Contenu de Glibc

(Dernière vérification effectuée auprès de la version 2.3.2.)

Glibc est une bibliothèque C qui apporte les appels système et les fonctions de base telles que open, malloc, printf, etc. La bibliothèque C est utilisée par tous les programmes liés dynamiquement.

Programmes installés : catchsegv, gencat, getconf, getent, glibcbug, iconv, iconvconfig, ldconfig, ldd, lddlibc4, locale, localedef, mtrace, nscd, nscd_nischeck, pcprofiledump, pt_chown, rpcgen, rpcinfo, sln, sprof, tzselect, xtrace, zdump et zic

Bibliothèques installées : ld.so, libBrokenLocale.[a,so], libSegFault.so, libanl.[a,so], libbsd−compat.a, libc.[a,so], libc_nonshared.a, libcrypt.[a,so], libdl.[a,so], libg.a, libieee.a, libm.[a,so], libmcheck.a, libmemusage.so, libnsl.a, libnss_compat.so, libnss_dns.so, libnss_files.so, libnss_hesiod.so, libnss_nis.so, libnss_nisplus.so, libpcprofile.so, libpthread.[a,so], libresolv.[a,so], librpcsvc.a, librt.[a,so], libthread_db.so et libutil.[a,so]

Descriptions courtes

catchsegv peut être utilisé pour créer une trace de la pile quand un programme s'est arrêté avec une erreur de segmentation.

gencat génère des catalogues de messages.

getconf affiche les valeurs de configuration du système pour les variables spécifiques au système de fichiers.

getent obtient des entrées de la base de données d'administration.

glibcbug crée un rapport de bogue et l'envoie par courrier électronique à l'adresses pour les bogues.

iconv réalise des conversions sur un ensemble de caractères.

iconvconfig crée un fichier de configuration du module iconv.

ldconfig configure les liaisons à l'exécution de l'éditeur de liens.

ldd indique quelles bibliothèques partagées sont requises pour chaque programme ou bibliothèque partagée donnée.

lddlibc4 assiste ldd avec les fichiers objets.

locale est un programme Perl indiquant au compilateur d'activer ou non l'utilisation des locales POSIX pour les opérations intégrées.

localedef compile les spécifications locales.

mtrace...

nscd est un démon cache DNS apportant un cache pour les requêtes DNS les plus courantes.

nscd_nischeck vérifie si, oui ou non, le mode sécurisé est nécessaire pour les recherches NIS+.

pcprofiledump affiche l'information générée par le profilage PC.

pt_chown est un programme d'aide pour grantpt pour initialiser les droits du propriétaire, du groupe et des autres à un pseudo−terminal esclave.

rpcgen génère du code C pour implémenter le protocole RPC.

rpcinfo fait un appel RPC à un serveur RPC.

sln est utilisé pour créer des liens symboliques. Le programme est lié statiquement, donc il est utile pour créer des liens symboliques vers des bibliothèques dynamiques si le système de liens dynamiques n'est pas fonctionnel pour quelque raison que ce soit.

sprof lit et affiche les données du profilage des objets partagés.

tzselect demande à l'utilisateur l'emplacement de son système et indique la description de zone horaire correspondante.

xtrace trace l'exécution d'un programme en affichant la fonction en cours d'exécution.

zdump est l'afficheur de la zone horaire.

zic est le compilateur de la zone horaire.

ld.so est le programme d'aide pour les exécutables de bibliothèques partagées.

libBrokenLocale est utilisé par des programmes, tels que Mozilla, pour résoudre les locales cassées.

libSegFault est un gestionnaire de signal d'erreur de segmentation. Il essaie de récupérer ces signaux.

libanl est une bibliothèque de recherche de noms asynchrone.

libbsd−compat apporte la portabilité nécessaire pour faire fonctionner certains programmes BSD sous Linux.

libc est la bibliothèque C principale −− une collection de fonctions communément utilisées.

libcrypt est la bibliothèque de cryptographie.

libdl est la bibliothèque d'interface pour l'édition des liens.

libg est une bibliothèque pour g++.

libieee est la bibliothèque pour les calculs en virgules flottantes IEEE.

libm est la bibliothèque mathématiques.

libmcheck contient du code lancé au démarrage.

libmemusage est utilisé par memusage pour aider à la récolte d'information sur l'utilisation de la mémoire par un programme.

libnsl est la bibliothèque des services réseau.

libnss* sont les bibliothèques "Name Service Switch", contenant des fonctions pour résoudre des noms d'hôtes, des noms d'utilisateurs, des noms de groupes, des alias, des services, des protocoles et ainsi de suite.

libpcprofile contient des fonctions de profilage utilisées pour tracer le temps CPU dépensé dans les lignes de code source.

libpthread est la bibliothèque des threads POSIX.

libresolv contient des fonctions pour créer, envoyer et interpréter des paquets provenant des serveurs de noms de domaines Internet.

librpcsvc contient des fonctions apportant différent services RPC.

librt contient des fonctions apportant la plupart des interfaces spécifiées par la "POSIX.1b Realtime Extension".

libthread_db contient des fonctions utiles pour construire des débogueurs de programmes multi−threadés.

libutil contient du code pour les fonctions "standard" utilisées dans beaucoup de différents utilitaires Unix.

Dépendances d'installation de Glibc

Glibc dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Gettext, Grep, Make, Perl, Sed, Texinfo.

Grep

Pour les instructions d'installation, voir la section intitulée *Installer Grep−2.5.1* dans Chapitre 6.

Site officiel de téléchargement

Grep (2.5.1): ftp://ftp.gnu.org/gnu/grep/

Contenu de Grep

Grep est un programme utilisé pour imprimer des lignes d'un fichier ressemblant à un modèle spécifié.

Programmes installés : egrep (lien vers grep), fgrep (lien vers grep) et grep

Descriptions courtes

egrep affiche les lignes des fichiers correspondant au motif d'une expression régulière étendue.

fgrep affiche les lignes des fichiers contenant une expression littérale.

grep affiche les lignes des fichiers correspondant au motif d'une expression régulière basique.

Dépendances d'installation de Grep

Grep dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Make, Sed, Texinfo.

Groff

Pour les instructions d'installation, voir la section intitulée *Installer Groff−1.19* dans Chapitre 6.

Site officiel de téléchargement

Groff (1.19): ftp://ftp.gnu.org/gnu/groff/

Contenu de Groff

Le package Groff inclut plusieurs programmes pour traiter du texte et le formatter. Groff traduit du texte et des commandes spécifiques en une sortie formattée, telle que ce que vous voyez dans une page man.

Programmes installés : addftinfo, afmtodit, eqn, eqn2graph, geqn (lien vers eqn), grn, grodvi, groff, groffer, grog, grolbp, grolj4, grops, grotty, gtbl (lien vers tbl), hpftodit, indxbib, lkbib, lookbib, mmroff, neqn, nroff, pfbtops, pic, pic2graph, post−grohtml, pre−grohtml, refer, soelim, tbl, tfmtodit, troff et zsoelim (lien vers soelim)

Descriptions courtes

addftinfo lit un fichier de police troff et ajoute quelques informations supplémentaires sur les mesures des polices, informations utilisées par le système groff.

afmtodit crée un fichier de police à utiliser avec groff et grops.

eqn compile les descriptions d'équations embarquées dans les fichiers d'entrée de troff pour les transformer en commandes compréhensibles par troff.

eqn2graph convertit une équation EQN en une image.

grn est un préprocesseur groff pour les fichiers gremlin.

grodvi est un pilote pour groff produisant un format dvi TeX.

groff est une interface pour le système de formattage des documents groff. Normalement, il lance le programme troff ainsi qu'un pré−processeur approprié pour le périphérique sélectionné.

groffer affiche des fichiers groff et des pages man sur des terminaux X et tty.

grog lit des fichiers et devine quelles options de groff parmi −e, −man, −me, −mm, −ms, −p, −s et −t sont nécessaires pour imprimer les fichiers, et indique la commande groff incluant ces options.

grolbp est un pilote groff pour les imprimantes Canon CAPSL (imprimantes laser de la série des LBP−4 et LBP−8).

grolj4 est un pilote pour groff produisant une sortie au format PCL5 compatible avec les imprimantes HP LaserJet 4.

grops traduit la sortie de GNU troff en Postscript.

grotty traduit la sortie de GNU troff en un format compatible pour les périphériques style machine à écrire.

gtbl est l'implémentation GNU de tbl.

hpftodit crée un fichier de police à utiliser avec groff −Tlj4 à partir d'un fichier de mesures de police pour HP.

indxbib crée un index renversé pour les bases de données bibliographiques dans un fichier spécifié à utiliser avec refer, lookbib et lkbib.

lkbib cherche dans les bases de données bibliographiques des références contenant des mots clés spécifiés et les affiche.

lookbib affiche une invite sur la sortie erreur standard (à moins que l'entrée standard ne soit un terminal), lit à partir de l'entrée standard une ligne contenant un ensemble de mots clés, recherche les bases de données bibliographiques dans un fichier spécifié pour les références contenant ces mots clés, affiche toute référence trouvée sur la sortie standard et répète ce processus jusqu'à la fin de l'entrée.

mmroff est un simple pré−processeur pour groff.

neqn formatte les équations pour une sortie ascii.

nroff est un script émulant la commande nroff utilisant groff.

pfbtops traduit une police Postscript au format .pfb en ASCII.

pic compile les descriptions d'images intégrées dans des fichiers d'entrées troff ou TeX et les enregistre en tant que commandes compréhensibles par TeX ou troff.

pic2graph convertit un diagramme PIC en une image.

pre−grohtml traduit la sortie de GNU troff en html.

post−grohtml traduit la sortie de GNU troff en html.

refer copie le contenu d'un fichier vers la sortie standard, à part que les lignes entre .[et .] sont interprétés comme des citations, et les lignes entre .R1 et .R2 en tant que commandes sur la façon de procéder avec les citations.

soelim lit des fichiers et rempalce les lignes de la forme .*so fichier* par le contenu du *fichier* mentionné.

tbl compile les descriptions des tables intégrées aux fichiers d'entrée de troff en commandes compréhensibles par troff.

tfmtodit crée un fichier de police à utiliser avec groff −Tdvi.

troff est hautement compatible avec troff Unix. Habituellement, il est appelé en utilisant la commande groff, qui lancera aussi des pré−processeurs et des post−processeurs dans l'ordre approprié et avec les options appropriées.

zsoelim est l'implémentation GNU de soelim.

Dépendances d'installation de Groff

Groff dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Grub

Pour des instructions d'installation, voir la section intitulée *Installer Grub−0.93* dans Chapitre 6.

Site officiel de téléchargement

Grub (0.93) : ftp://alpha.gnu.org/pub/gnu/grub/ Correctif "Grub Gcc33"
http://www.linuxfromscratch.org/patches/lfs/5.0/grub−0.93−gcc33−1.patch

Contenu de Grub

Le package Grub contient un chargeur de démarrage.

Programmes installés : grub, grub−install, grub−md5−crypt, grub−terminfo et mbchk

Descriptions courtes

grub est le shell de commandes pour le chargeur de démarrage de GRUB (GRand Unified Bootloader).

grub−install installe GRUB sur le périphérique donné.

grub−md5−crypt crypte un mot de passe au format MD5.

grub−terminfo génère une commande terminfo à partir d'un nom name. Il peut être utilisé si vous avez un terminal peu commun.

mbchk vérifie le format d'un noyau multiboot.

Dépendances d'installation de Grub

Grub dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Gzip

Pour les instructions d'installation, voir la section intitulée *Installer Gzip−1.3.5* dans Chapitre 6.

Site officiel de téléchargement

Gzip (1.3.5): ftp://alpha.gnu.org/gnu/gzip/

Contenu de Gzip

Le package Gzip contient des programmes pour compresser et décompresser des fichiers utilisant le codage Lempel−Ziv (LZ77).

Programmes installés : gunzip (lien vers gzip), gzexe, gzip, uncompress (lien vers gunzip), zcat (lien vers gzip), zcmp, zdiff, zegrep, zfgrep, zforce, zgrep, zless, zmore et znew

Descriptions courtes

gunzip décompresse les fichiers gzip.

gzexe est utilisé pour créer des fichiers exécutables auto−extractibles.

gzip compresse les fichiers donnés, en utilisant le codage Lempel−Ziv (LZ77).

zcat décompresse les fichiers gzip donnés vers la sortie standard.

zcmp lance cmp sur des fichiers gzip.

zdiff lance diff sur des fichiers gzip.

zegrep lance egrep sur des fichiers gzip.

zfgrep lance fgrep sur des fichiers gzip.

zforce force une extension .gz sur tous les fichiers donnés étant des fichiers gzip, de façon à ce que gzip ne les compresse pas de nouveau. Ceci peut être utile lorsque des noms de fichiers sont tronqués lors d'un transfert de fichiers.

zgrep lance grep sur des fichiers gzip.

zless lance less sur des fichiers gzip.

zmore lance more sur des fichiers gzip.

znew recompresse des fichiers au format compress vers le format gzip −− .Z en .gz.

Dépendances d'installation de Gzip

Gzip dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Inetutils

Pour les instructions d'installation, voir la section intitulée *Installer Inetutils−1.4.2* dans Chapitre 6.

Site de téléchargement officiel

Inetutils (1.4.2): http://freshmeat.net/projects/inetutils/

Contenu de Inetutils

Le package Inetutils contient des clients et des serveurs réseau.

Programmes installés : ftp, ping, rcp, rlogin, rsh, talk, telnet et tftp

Descriptions courtes

ftp est le programme de transfert de fichier ARPANET.

ping envoie des paquets echo−request et indique le temps qu'a pris la réponse.

rcp copie des fichiers à distance.

rlogin permet les connexions à distance.

rsh lance un shell distant.

talk est utilisé pour discuter avec un autre utilisateur.

telnet est une interface pour le protocole TELNET.

tftp est un programme trivial de transfert de fichiers.

Dépendance d'installation d'Inetutils

Inetutils dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Kbd

Pour les instructions d'installation, voir la section intitulée *Installer Kbd−1.08* dans Chapitre 6.

Site officiel de téléchargement

Kbd (1.08) : ftp://ftp.win.tue.nl/pub/linux−local/utils/kbd/ Correctif "Kbd More−Programs" : http://www.linuxfromscratch.org/patches/lfs/5.0/kbd−1.08−more−programs.patch

Contenu de Kbd

Kbd contient des fichiers de codage des touches et des utilitaires pour le clavier.

Programmes installés : chvt, deallocvt, dumpkeys, fgconsole, getkeycodes, getunimap, kbd_mode, kbdrate, loadkeys, loadunimap, mapscrn, openvt, psfaddtable (lien vers psfxtable), psfgettable (lien vers psfxtable), psfstriptable (lien vers psfxtable), psfxtable, resizecons, setfont, setkeycodes, setleds, setlogcons, setmetamode, setvesablank, showconsolefont, showkey, unicode_start et unicode_stop

Descriptions courtes

chvt modifie le terminal virtual courant.

deallocvt désalloue les terminaux virtuels inutilisés.

dumpkeys retrouve les tables de traduction du clavier.

fgconsole affiche le nombre de terminaux virtuels actifs.

getkeycodes affiche la table de correspondance scancode vers keycode du noyau.

getunimap affiche l'unimap actuellement utilisé.

kbd_mode indique ou initialise le mode du clavier.

kbdrate initialise les taux de répétition et de délai du clavier.

loadkeys charge les tables de traduction du clavier.

loadunimap charge la table de correspondance unicode vers font du noyau.

mapscrn est un programme obsolète qui chargeait une table de correspondance des caractères en sortie définie par l'utilisateur pour le pilote de la console. Ceci est maintenant effectué par setfont.

openvt lance un programme sur un nouveau terminal virtuel (VT).

psf* sont un ensemble d'outils pour gérer les tables de caractères Unicode pour les polices de la console.

resizecons modifie l'idée du noyau sur la taille de la console.

setfont vous laisse modifier les polices EGA/VGA sur la console.

setkeycodes charge les entrées de la table de correspondance scancode vers keycode du noyau, utile si vous avez quelques touches inhabituelles sur votre clavier.

setleds initialise le clavier et les LED. Beaucoup de personnes le trouvent utile pour alllumer NumLock par défaut, setleds +num fait cela.

setlogcons envoie des messages du noyau vers la console.

setmetamode definit la gestion des touches supplémentaires du clavier.

setvesablank vous laisse configurer finement la sauvegarde d'écran matérielle (pas de grille−pains, juste un écran blanc).

showconsolefont affiche les propriétés actuelles de la police de caractères de la console EGA/VGA.

showkey rapporte les scancodes et keycodes ainsi que les codes ASCII des touches appuyées sur le clavier.

unicode_start place le clavier et la console dans un mode unicode.

unicode_stop supprime le clavier et la console du mode unicode.

Dépendances d'installation de Kbd

Kbd dépend de Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Gettext, Glibc, Grep, Gzip, M4, Make, Sed.

Less

Pour les instructions d'installation, voir la section intitulée *Installer Less−381* dans Chapitre 6.

Site officiel de téléchargement

Less (381): ftp://ftp.gnu.org/gnu/less/

Contenu de Less

Le programme less est un paginateur de fichier (ou un afficheur de texte). Il affiche le contenu d'un fichier et permet le défilement du texte. Less dispose de quelques fonctionnalités qui ne sont pas incluses dans le paginateur << more >>, telle que le défilement arrière.

Programmes installés : less, lessecho et lesskey

Descriptions courtes

less est un visualisateur de fichier page par page. Il affiche le contenu d'un fichier donné, vous laisser avancer ou reculer, trouver un texte et sauter vers des marques.

lessecho est nécessaire pour étendre des métacaractères, tels que * et ? dans les noms de fichiers sur des systèmes Unix.

lesskey est utilisé pour spécifier les touches de fonctions de less.

Dépendances d'installation de Less

Less dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

LFS−Bootscripts

Pour les instructions d'installation, voir la section intitulée *Installer LFS−Bootscripts−1.12* dans Chapitre 6.

Site officiel de téléchargement

LFS−Bootscripts (1.12): http://downloads.linuxfromscratch.org/

Contenu de LFS−bootscripts

Le package LFS−Bootscripts contient des scripts shell style init SysV. Ces scripts réalisent plusieurs tâches telles que la vérification de l'intégrité d'un système de fichiers lors du redémarrage, le chargement du plan de codage du clavier, la configuration du réseau et l'arrêt des processus lors de l'arrêt du système.

Scripts installés : checkfs, cleanfs, functions, halt, ifdown, ifup, loadkeys, localnet, mountfs, mountproc, network, rc, reboot, sendsignals, setclock, swap, sysklogd et template

Descriptions courtes

Le script **checkfs** vérifie les systèmes de fichiers juste avant qu'ils ne soient montés (avec une exception pour les systèmes de fichiers journalisés ou réseau).

Le script **cleanfs** supprime les fichiers qui ne devraient pas être conservés entre les redémarrages, tels que ceux compris dans `/var/run/` et `/var/lock/`. Il re−crée de nouveau `/var/run/utmp` et supprime les fichiers probables `/etc/nologin`, `/fastboot` et `/forcefsck`.

Le script **functions** contient des fonctions partagées entre différents scripts, tels que les vérifications d'erreur et de statut.

Le script **halt** arrête le système.

Les scripts **ifdown** et **ifup** portent assistance au script network avec les périphériques réseau.

Le script **loadkeys** charge la table de codage du clavier que vous avez spécifiez.

Le script **localnet** initialise le nom d'hôte du système et le périphérique loopback local.

Le script **mountfs** monte tous les systèmes de fichiers qui ne sont pas marqués noauto ou qui ne sont pas réseau.

Le script **mountproc** est utilisé pour monter le système de fichiers proc.

Le script **network** initialise les interfaces réseau, telles que les cartes réseau, et initialise la passerelle par défaut lorsque cela est applicable.

Le script **rc** est le script de contrôle du niveau d'exécution maître. Il est responsable du lancement de tous les autres scripts un par un dans une séquence spécifique.

Le script **reboot** relance le système.

Le script **sendsignals** s'assure que chaque processus soit terminé avant que le système ne se relance ou ne s'arrête.

Le script **setclock** réinitialise l'horloge du noyau avec l'heure locale au cas où l'horloge matérielle n'est pas sur l'heure GMT.

Le script **swap** active et désactive les partitions swap.

Le script **sysklogd** lance et arrêt les démons des traces système et noyau.

Le script **template** est un modèle que vous pouvez utiliser pour créer vos propres scripts de démarrage pour vos autres démons.

Dépendances d'installation de Bootscripts

Bzip2 dépend de Bash, Coreutils.

Lfs−Utils

Pour les instructions d'installation, voir la section intitulée *Installer Lfs−Utils−0.3* dans Chapitre 6.

Site officiel de téléchargement

Lfs−utils (0.3): http://www.linuxfromscratch.org/~winkie/downloads/lfs−utils/

Contenu de Lfs−Utils

Le package Lfs−Utils contient quelques programmes divers provenant de plusieurs packages mais qui ne sont pas assez importants pour disposer de leur propre package.

Programmes installés : mktemp, tempfile, http−get et iana−net

Fichiers installés : protocols, services

Descriptions courtes

mktemp crée des fichiers temporaires d'une façon sécurisée.

tempfile crée des fichiers temporaires d'une manière moins sécurisée que **`mktemp`**. Il est installé pour des raisons de compatibilité descendante.

Le script **http−get** prend avantage d'une fonctionnalité peu connue de **`bash`** appelée la redirection du réseau ("net redirection"). Il est utilisé pour télécharger à partir de sites web sans utiliser d'autres programmes.

iana−net utilise le script **`http−get`** pour simplifier le processus de récupération des fichiers de configuration services et protocols.

Dépendances d'installation de Lfs−Utils

(Les dépendances n'ont pas encore été vérifié.)

Libtool

Pour les instructions d'installation, voir la section intitulée *Installer Libtool−1.5* dans Chapitre 6.

Site officiel de téléchargement

Libtool (1.5) : ftp://ftp.gnu.org/gnu/libtool/

Contenu de Libtool

GNU libtool est un outil générique de support de scripts. Libtool cache la complexité en utilisant les bibliothèques partagées derrière une interface portable cohérente.

Programmes installés : libtool et libtoolize

Bibliothèques installées : libltdl.[a,so]

Descriptions courtes

libtool apporte des services généralisés de support pour la construction de bibliothèques.

libtoolize fournit un moyen standard d'ajouter le support de libtool à un package.

libltdl cache les multiples difficultés posées par les bibliothèques dlopening (NdT : ouverture de bibliothèque partagée).

Dépendances d'installation de Libtool

Libtool dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Linux (le noyau)

Pour les instructions d'installation, voir la section intitulée *Installer Linux−2.4.22* dans Chapitre 8.

Site officiel de téléchargement

Linux (2.4.22): ftp://ftp.kernel.org/pub/linux/kernel/

Contenu de Linux

Le noyau Linux kernel est au coeur de chaque système Linux. C'est ce qui fait tourner Linux. Lorsqu'un ordinateur est allumé et lance un système Linux, la première pièce de logiciel Linux à être chargé est le noyau.

Celui−ci initialise les composants matériels du système: ports séries, ports parallèles, cartes son, cartes réseaux, contrôleurs IDE, contrôleurs SCSI et beaucoup plus encore. EN bref, le noyau rend le matériel disponible aux logiciels exécutés.

Fichiers installés : le noyau et ses en−têtes.

Descriptions courtes

Le *noyau* est le moteur de votre système GNU/Linux. Au lancement de votre machine, le noyau est la première partie de votre système d'exploitation à être chargée. Il détecte et initialise tous les composants matériels de votre ordinateur, puis rend ces composants disponibles au logiciel sous forme d'un arbre de fichiers et transforme un simple CPU en une machine multi−tâches capables de réaliser les tâches de différents programmes pratiquement en même temps.

Les *entêtes du noyau* définissent l'interface vers les services que le noyau apporte. Les entêtes dans le répertoire `include` de votre système devrait *toujours* être ceux avec lesquels Glibc a été compilé et ne devraient donc *pas* être remplacés lors de la mise à jour du noyau.

Dépendances d'installation de Linux

Linux dépend de Bash, Binutils, Coreutils, Findutils, GCC, Glibc, Grep, Gzip, Make, Modutils, Perl, Sed.

M4

Pour les instructions d'installation, voir la section intitulée *Installer M4−1.4* dans Chapitre 6.

Site officiel de téléchargement

M4 (1.4): ftp://ftp.gnu.org/gnu/m4/

Contenu de M4

M4 est un processeur de macro. Il copie l'entrée vers la sortie, exécutant les macros qu'il trouve. Les macros sont soit intégrées soit définies par l'utilisateur, et peuvent prendre n'importe quel nombre d'arguments. En plus d'exécuter certaines macros, m4 dispose de fonctions intégrées permettant d'inclure des fichiers nommés, de lancer des commandes Unix, de réaliser quelques calculs arithmétiques, de manipuler du texte de différentes façons, d'utiliser la récursivité, etc... Le programme m4 est utilisé soit comme interface pour un compilateur soit directement comme un processeur de macros.

Programme installé : m4

Descriptions courtes

m4 copie les fichiers donnés lors de l'expansion des macros qu'il contient. Ces macros sont soit intégrées soit définies par l'utilisateur et peuvent prendre un certain nombre d'arguments. En plus de la simple expansion de macros, m4 dispose de fonctions intégrées pour inclure des fichiers nommés, lancer des commandes Unix,

faire des calculs arithmétiques, manipuler du texte de différentes façons, faire de la récursion et plus encore. Le programme m4 peut être utilisé soit comme interface à un compilateur soit comme un processeur pour macros.

Dépendances d'installation de M4

M4 dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Perl, Sed.

Make

Pour les instructions d'installation, voir la section intitulée *Installer Make−3.80* dans Chapitre 6.

Site officiel de téléchargement

Make (3.80): ftp://ftp.gnu.org/gnu/make/

Contenu de Make

Make détermine, automatiquement, quelles pièces d'un grand programme ont besoin d'être recompilées et envoie les commandes pour les recompiler.

Programme installé : make

Descriptions courtes

make détermine automatiquement quelles pièces d'un gros package ont besoin d'être recompilé, puis de lancer les commandes adéquates.

Dépendances d'installation de Make

Make dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Sed.

MAKEDEV

Pour les instructions d'installation, voir la section intitulée *Créer les périphériques (Makedev−1.7)* dans Chapitre 6.

Site officiel de téléchargement

MAKEDEV (1.7) http://downloads.linuxfromscratch.org/

Contenu de MAKEDEV

Le script MAKEDEV crée les noeuds de périphériques statiques qui résident habituellement dans le répertoire `/dev`. Des informations détaillées sur les noeuds périphériques puissent être trouvées dans le fichier

`Documentation/devices.txt` dans le répertoire des sources du noyau Linux.

Script installé : MAKEDEV

Descriptions courtes

MAKEDEV est un script créant les fichiers périphériques statiques nécessaires, résidant habituellement dans le répertoire `/dev`.

Dépendances d'installation de MAKEDEV

Make dépend de Bash, Coreutils.

Man

Pour les instructions d'installation, voir la section intitulée *Installer Man−1.5m2* dans Chapitre 6.

Site officiel de téléchargement

Man (1.5m2): ftp://ftp.win.tue.nl/pub/linux−local/utils/man/ Correctif "Man 80−Columns" : http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−80cols.patch Correctif "Man Manpath" : http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−manpath.patch Correctif "Man Pager" : http://www.linuxfromscratch.org/patches/lfs/5.0/man−1.5m2−pager.patch

Contenu de Man

Man est un afficheur de pages man.

Programmes installés : apropos, makewhatis, man, man2dvi, man2html et whatis

Descriptions courtes

apropos recherche dans la base de données whatis et affiche des descriptions courtes des commandes systèmes contenant une chaîne particulière.

makewhatis construit la base de données whatis. Il lit toutes les pages man contenues dans manpath et, pour chaque page, écrit le nom et une courte description dans la base de données whatis.

man formatte et affiche la page man demandée.

man2dvi convertit une page man au format dvi.

man2html convertit une page man au format html.

whatis recherche dans la base de données whatis et affiche les courtes descriptions des commandes système contenant un certain mot clé en tant que mot séparé.

Dépendances d'installation de Man

Man dépend de Bash, Binutils, Coreutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Man−pages

Pour les instructions d'installation, voir la section intitulée *Installer Man−pages−1.60* dans Chapitre 6.

Site officiel de téléchargement

Man−pages (1.60): ftp://ftp.kernel.org/pub/linux/docs/manpages/

Contenu de Man−pages

Le package Man−pages contient plus de 1200 pages man. Cette documentation détaille les fonctions C et C++, décrit quelques fichiers périphériques importants et apporte les documents qui seraient autrement être manquants des autres packages.

Fichiers installés : différentes pages man

Descriptions courtes

Les exemples de *pages de manuel* fournis décrivent toutes les fonctions C et C++, quelques fichiers périphériques importants et autres fichiers de configuration importants.

Dépendances d'installation de Man−pages

Man dépend de Bash, Coreutils, Make.

Modutils

Pour les instructions d'installation, voir la section intitulée *Installer Modutils−2.4.25* dans Chapitre 6.

Site officiel de téléchargement

Modutils (2.4.25): ftp://ftp.kernel.org/pub/linux/utils/kernel/modutils/

Contenu de Modutils

Le package Modutils contient des programmes que vous pouvez utiliser pour gérer les modules du noyau.

Programmes installés : depmod, genksyms, insmod, insmod_ksymoops_clean, kallsyms (lien vers insmod), kernelversion, ksyms (lien vers insmod), lsmod (lien vers insmod), modinfo, modprobe (lien vers insmod) et rmmod (lien vers insmod)

Descriptions courtes

depmod crée un fichier de dépendances, basé sur les symbôles qu'il trouve dans l'ensemble de modules existant. Ce fichier de dépendances est utilisé par modprobe pour charger automatiquement les modules requis.

genksyms génère une information de version des symbôles.

insmod installe un module chargeable dans le noyau en cours d'exécution.

insmod_ksymoops_clean supprime les ksyms sauvés et les modules non accédés pendant deux jours.

kallsyms extrait les symbôles du noyau pour le débuggage.

kernelversion rapporte la version majeure du noyau en cours d'exécution.

ksyms affiche les symbôles exportés du noyau.

lsmod affiche les modules chargés.

modinfo examine un fichier objet associé avec un module du noyau et affiche toute information qu'il peut trouver.

modprobe utilise un fichier de dépendances, créé par depmod, pour charger automatiquement les modules concernés.

rmmod décharge les modules du noyau en cours d'exécution.

Dépendances d'installation de Modutils

Modutils dépend de Bash, Binutils, Bison, Coreutils, Diffutils, Flex, GCC, Glibc, Grep, M4, Make, Sed.

Ncurses

Pour les instructions d'installation, voir la section intitulée *Installer Ncurses−5.3* dans Chapitre 6.

Site officiel de téléchargement

Ncurses (5.3) : ftp://ftp.gnu.org/gnu/ncurses/ Correctif "Ncurses Etip" : http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−etip−2.patch Correctif "Ncurses Vsscanf" : http://www.linuxfromscratch.org/patches/lfs/5.0/ncurses−5.3−vsscanf.patch

Contenu de Ncurses

(Dernière version effectuée auprès de la version 5.3.)

Le package Ncurses apporte des bibliothèques de gestion de caractères et de terminaux, incluant les panneaux et les menus.

Programmes installés : captoinfo (lien vers tic), clear, infocmp, infotocap (lien vers tic), reset (lien vers tset), tack, tic, toe, tput et tset

Bibliothèques installées : libcurses.[a,so] (lien vers libncurses.[a,so]), libform.[a,so], libmenu.[a,so], libncurses++.a, libncurses.[a,so], libpanel.[a,so]

Descriptions courtes

captoinfo convertit une description termcap en une description terminfo.

clear efface l'écran si c'est possible.

infocmp compare ou affiche des descriptions terminfo.

infotocap convertit une description terminfo en description termcap.

reset ré−initialise un terminal à ses valeurs par défaut.

tack est le vérificateur d'actions pour terminfo. Il est principalement utilisé pour tester la validité d'une entrée dans la base de données terminfo.

tic est le compilateur entrée−description de terminfo. Il traduit un fichier terminfo à partir du format source en un format binaire nécessaire pour les routines de la bibliothèque ncurses. Un fichier terminfo contient des informations sur les capacités d'un certain terminal.

toe liste tous les types de terminaux disponibles, avec pour chacun d'entre eux son nom principal et sa description.

tput rend disponible les valeurs des capacités d'un terminal au shell. Il peut aussi être utilisé pour ré−initialiser ou pour simplement initialiser un terminal, ou pour rapporter son nom long.

tset peut être utilisé pour initialiser des terminaux.

libncurses* contient des fonctions pour afficher le texte de plusieurs façons très complexes sur l'écran d'un terminal. Un bon exemple de l'utilisation de ces fonctions est le menu affiché lors du make menuconfig du noyau.

libform* contient des fonctions pour implémenter des formes.

libmenu* contient des fonctions pour implémenter des menus.

libpanel* contient des fonctions pour implémenter des panneaux.

Dépendances d'installation de Ncurses

Ncurses dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Net−tools

Pour les instructions d'installation, voir la section intitulée *Installer Net−tools−1.60* dans Chapitre 6.

Site officiel de téléchargement

Net−tools (1.60) : http://www.tazenda.demon.co.uk/phil/net−tools/ Correctif "Net−tools Mii−Tool−Gcc33" : http://www.linuxfromscratch.org/patches/lfs/5.0/net−tools−1.60−miitool−gcc33−1.patch

Contenu de Net−tools

Le package Net−tools contient une collection de programmes formant la base réseau sous Linux.

Programmes installés : arp, dnsdomainname (lien vers hostname), domainname (lien vers hostname), hostname, ifconfig, nameif, netstat, nisdomainname (lien vers hostname), plipconfig, rarp, route, slattach et ypdomainname (lien vers hostname)

Descriptions courtes

arp est utilisé pour manipuler le cache ARP du noyau, habituellement pour ajouter ou supprimer une entrée, ou pour sortir le cache entier.

dnsdomainname rapporte le nom de domaine du DNS du système.

domainname rapporte ou initialise le nom de domaine NOS/YP du système.

hostname rapporte ou initialise le nom de l'hôte système actuel.

ifconfig est le principal utilitaire pour configurer les interfaces réseau.

nameif nomme les interfaces réseau basées sur les adresses MAC.

netstat est utilisé pour rapporter les connexions réseau, les tables de routage et les statistiques des interfaces.

nisdomainname fait la même chose que domainname.

plipconfig est utilisé pour configurer finement les paramètres du périphérique PLIP, pour améliorer ses performances.

rarp est utilisé pour manipulater la table RARP du noyau.

route est utilisé pour manipulater la table de routage IP.

slattach attache une interface réseau à une ligne série. Ceci vous permet d'utiliser les lignes normales de terminaux pour des liens en point−à−point vers d'autres ordinateurs.

ypdomainname fait la même chose que domainname.

Dépendances d'installation de Net−tools

Net−tools dépend de Bash, Binutils, Coreutils, GCC, Glibc, Make.

Patch

Pour les instructions d'installation, voir la section intitulée *Installer Patch−2.5.4* dans Chapitre 6.

Site officiel de téléchargement

Patch (2.5.4): ftp://ftp.gnu.org/gnu/patch/

Contenu de Patch

Le programme patch modifie un fichier suivant un fichier patch, appelé aussi correctif. Un correctif est habituellement une liste, créée par le programme diff, et contenant les instructions sur la façon dont le fichier original a été modifié.

Programme installé : patch

Descriptions courtes

patch modifie des fichiers suivant un fichier de corrections (appelé correctif ou patch). Habituellement, un correctif est une liste de différences créée par le programme diff. En appliquant ces différences aux fichiers originaux, patch crée les versions corrigées. Utiliser des correctifs plutôt qu'une nouvelle archive tar complète pour garder vos sources à jour peut vous faire gagner énormément en temps de téléchargement.

Dépendances d'installation de Patch

Patch dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Perl

Pour les instructions d'installation, voir la section intitulée *Installer Perl−5.8.0* dans Chapitre 6.

Site officiel de téléchargement

Perl (5.8.0) : http://www.perl.com/ Correctif "Perl Libc" : http://www.linuxfromscratch.org/patches/lfs/5.0/perl−5.8.0−libc−3.patch

Contenu de Perl

Le package Perl contient perl, l'acronyme de Practical Extraction and Report Language (Langage pratique d'extraction et de rapport). Perl combine certaines des meilleurs fonctionnalités des langages C, sed, awk et sh en un seul langage extrêmement puissant.

Programmes installés : a2p, c2ph, dprofpp, enc2xs, find2perl, h2ph, h2xs, libnetcfg, perl, perl5.8.0 (lien vers perl), perlbug, perlcc, perldoc, perlivp, piconv, pl2pm, pod2html, pod2latex, pod2man, pod2text, pod2usage, podchecker, podselect, psed (lien vers s2p), pstruct (lien vers c2ph), s2p, splain and xsubpp

Bibliothèques installées : (trop nombreux pour les nommer toutes)

Descriptions courtes

a2p traduit awk en perl.

c2ph affiche les structures C générées à partir de "cc −g −S".

dprofpp affiche les données de profilage avec perl.

en2cxs construit une extension Perl pour le module Encode, soit à partir des "Unicode Character Mappings" soit à partir des fichiers de codage Tcl.

find2perl traduit les commandes find en perl.

h2ph convertit les fichiers d'entête .h C en fichier d'entête .ph Perl.

h2xs convertit les fichiers d'entête .h C en extensions Perl.

libnetcfg s'utilise pour configurer libnet.

perl combine quelques unes des meilleures fonctionnalités du C, de sed, d'awk et de sh en un seul langage semblable à couteau suisse.

perlbug est utilisé pour générer des rapports de bogues sur Perl ou sur les modules qui viennent avec lui, et les envoyer par courrier électronique.

perlcc génère des exécutables à partir des programmes Perl.

perldoc affiche une partie de la documentation au format pod qui est embarqué dans les répertoires d'installation de perl ou dans un script perl.

perlivp est l'acronyme de "Perl Installation Verification Procedure" (NdT : Procédure de Vérification d'Installation de Perl). Il est utilisé pour vérifier que Perl et ses bibliothèques sont bien installés.

piconv est une version Perl du convertisseur de codage des caractères **`iconv`**.

pl2pm est un outil brut pour convertir des fichiers Perl4 .pl en modules Perl5 .pm.

pod2html convertit des fichiers à partir du format pod au format HTML.

pod2latex convertit des fichiers à partir du format pod au format LaTeX.

pod2man convertit des données pod en entrée formattée *roff.

pod2text convertit des données pod en texte ANSI formatté.

pod2usage affiche le message d'usage à partir de documents pod embarqué dans des fichiers.

podchecker vérifie la syntaxe du format des fichiers de documentation pod.

podselect affiche les sections sélectionnées de la documentation pod.

psed est une version Perl de l'éditeur en flux **sed**.

pstruct affiche les structures C générées à partir de "cc −g −S".

s2p traduit sed en perl.

splain est utilisé pour forcer les diagnostiques à être verbeux dans perl.

xsubpp convertit le code Perl XS en code C.

Dépendances d'installation de Perl

Perl dépend de Bash, Binutils, Coreutils, Diffutils, Gawk, GCC, Glibc, Grep, Make, Sed.

Procinfo

Pour les instructions d'installation, voir la section intitulée *Installer Procinfo−18* dans Chapitre 6.

Site officiel de téléchargement

Procinfo (18) : ftp://ftp.cistron.nl/pub/people/svm/

Contenu de Procinfo

Le programme procinfo récupère les données système, telles que l'usage de la mémoire et les numéros d'IRQ, à partir du répertoire `/proc` et formatte ces données d'une façon conséquente.

Programmes installés : lsdev, procinfo et socklist

Descriptions courtes

lsdev liste les périphériques présents sur votre système ainsi que les IRQ et ports d'entrées qu'ils utilisent.

procinfo affiche un résumé des informations présentes dans le système de fichiers virtuel proc.

socklist liste les sockets ouvertes, rapportant leur type, numéro de port et autres spécificités.

Dépendances d'installation de Procinfo

Procinfo dépend de Binutils, GCC, Glibc, Make, Ncurses.

Procps

Pour les instructions d'installation, voir la section intitulée *Installer Procps−3.1.11* dans Chapitre 6.

Site officiel de téléchargement

Procps (3.1.11): http://procps.sourceforge.net/ Correctif "Procps Locale" : http://www.linuxfromscratch.org/patches/lfs/5.0/procps−3.1.11−locale−fix.patch

Contenu de Procps

Le package Procps apporte les programmes pour observer et arrêter les processus systèmes. Procps récupère des informations sur les processus via le répertoire `/proc`.

Programmes installés : free, kill, pgrep, pkill, pmap, ps, skill, snice, sysctl, tload, top, uptime, vmstat, w et watch

Bibliothèque installée : libproc.so

Descriptions courtes

free rapporte la quantité de mémoire, physique et swap, libre et utilisée dans le système.

kill est utilisé pour envoyer des signaux aux processus.

pgrep recherche des processus suivant leur nom et d'autres attributs.

pkill envoie un signal aux processus suivant leur nom et d'autres attributes.

pmap récupère le plan mémoire du processus donné.

ps affiche un état des processus actuels.

skill envie des signaux aux processus correspondant à un certain critère.

snice change la priorité des processus suivant un critère donné.

sysctl modifie les paramètres du noyau pendant son exécution.

tload affiche un graphe de la charge système actuelle.

top affiche les processus suivant le top CPU. Il apporte une interface indiquant l'activité du processeur en temps réel.

uptime affiche depuis combien de temps le système est allumé, combien d'utilisateurs sont connectés ainsi que la moyenne de la charge système.

vmstat rapporte les statistiques sur la mémoire virtuelle, donnant des information sur les processus, la mémoire, la pagination, les blocs d'entrée/sortie, les traps et l'activité du CPU.

w affiche les utilisateurs actuellement connectés, où et depuis.

watch lance une commande donnée de façon répétée, en affichant une permier écran de sa sortie. Ceci vous permet de vérifier le changement à la sortie tout le long.

libproc contient les fonctions utilisées par la plupart des programmes de ce package.

Dépendances d'installation de Procps

Procps dépend de Bash, Binutils, Coreutils, GCC, Glibc, Make, Ncurses.

Psmisc

Pour les instructions d'installation, voir la section intitulée *Installer Psmisc−21.3* dans Chapitre 6.

Site officiel de téléchargement

Psmisc (21.3): http://download.sourceforge.net/psmisc/
ftp://download.sourceforge.net/pub/sourceforge/psmisc/

Contenu de Psmisc

Le package Psmisc contient trois programmes qui aident à gérer le répertoire `/proc`.

Programmes installés : fuser, killall et pstree

Descriptions courtes

fuser rapporte le PID des processus utilisant les fichiers ou systèmes de fichiers indiqués.

killall tue les processus suivant leur nom. Il envoie un signal à tous les processus exécutant une des commandes données.

pidof rapporte le PID des programmes indiqués. (Notez que le programme pidof utilisé est celui de Sysvinit.)

pstree affiche les processus en cours avec une hiérarchie de type arbre.

Dépendances d'installation de Psmisc

Psmisc dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed.

Sed

Pour les instructions d'installation, voir la section intitulée *Installer Sed−4.0.7* dans Chapitre 6.

Site officiel de téléchargement

Sed (4.0.7): ftp://ftp.gnu.org/gnu/sed/

Contenu de Sed

sed est un éditeur en flux. Un éditeur en flux est utilisé pour réaliser des transformations de texte basique sur une entrée en flux (un fichier ou une entrée à partir d'un tuyau).

Programme installé : sed

Descriptions courtes

sed est utilisé pour filtrer et transformer des fichiers texte en une seule passe.

Dépendances d'installation de Sed

(Dernière vérification effectuée auprès de la version 3.02.)

Sed dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Texinfo.

Shadow

Pour les instructions d'installation, voir la section intitulée *Installer Shadow−4.0.3* dans Chapitre 6.

Site officiel de téléchargement

Shadow (4.0.3) : ftp://ftp.pld.org.pl/software/shadow/ Correctif "Shadow Newgrp" : http://www.linuxfromscratch.org/patches/lfs/5.0/shadow−4.0.3−newgrp−fix.patch

Contenu de Shadow

Le package Shadow a été créé pour augmenter la sécurité des mots de passe du système.

Programmes installés : chage, chfn, chpasswd, chsh, dpasswd, expiry, faillog, gpasswd, groupadd, groupdel, groupmod, groups, grpck, grpconv, grpunconv, lastlog, login, logoutd, mkpasswd, newgrp, newusers, passwd, pwck, pwconv, pwunconv, sg (lien vers newgrp), useradd, userdel, usermod, vigr (lien vers vipw) et vipw

Descriptions

chage est utilisé pour modifier le nombre maximum de jour entre lequel aura lieu les changements de mot de passe.

chfn est utilisé pour changer le nom d'un utilisateur et quelques autres informations.

chpasswd est utilisé pour mettre à jour les mots de passe d'une série complète de compte utilisateurs en un coup.

chsh est utilisé pour modifier le shell de connexion par défaut d'un utilisateur.

dpasswd est utilisé pour modifier les mots de passe distants pour les shells de connexion d'un utilisateur.

expiry vérifie et force la politique d'expiration des mots de passe.

faillog est utilisé pour examiner les traces de connexion échouées, pour initialiser un nombre maximum d'échecs avec le blocage d'un compte ou la réinitialisation le comptage des éches.

gpasswd est utilisé pour ajouter et supprimer des membres et administrateurs aux groupes.

groupadd crée un groupe avec le nom donné.

groupdel supprime le groupe avec le nom donné.

groupmod est utilisé pour modifier le nom donné du groupe ou son GID.

groups indique les groupes auxquels appartiennent les utilisateurs donnés.

grpck vérifie l'intégrité des fichiers groupes, `/etc/group` et `/etc/gshadow`.

grpconv crée ou met à jour le fichier groupe shadow à partir du fichier groupe normal.

grpunconv met à jour `/etc/group` à partir de `/etc/gshadow` puis supprime ce dernier.

lastlog indique la connexion la plus récente de chaque utilisateur ou d'un utilisateur spécifié.

login est utilisé par le système pour permettre aux utilisateurs de se connecter.

logoutd est un démon utilisé pour renforcer les restrictions sur les temps de connexion et les ports.

mkpasswd crypte le mot de passe donné en utilisant la perturbation, elle−aussi spécifiée.

newgrp est utilisé pour changer le GID actuel lors d'une session.

newusers est utilisé pour créer ou mettre à jour toute une série de comptes utilisateurs en une seule fois.

passwd est utilisé pour modifier le mot de passe d'un compte utilisateur ou groupe.

pwck vérifie l'intégrité des fichiers mot de passe, `/etc/passwd` et `/etc/shadow`.

pwconv crée ou met à jour le fichier de mots de passe shadow à partir du fichier de mots de passe normal.

pwunconv met à jour `/etc/passwd` à partir de `/etc/shadow` puis supprime ce dernier.

sg exécute une commande donné alors que le GID de l'utilisateur est initialisé à celui du groupe indiqué.

useradd crée un nouvel utilisateur avec le nom donné, ou met à jour l'information du nouvel utilisateur par défaut.

userdel supprime le compte utilisateur donné.

usermod est utilisé pour modifier le login, l'UID, le shell, le groupe initial, le répertoire personnel de l'utilisateur indiqué.

vigr peut être utilisé pour éditer les fichiers `/etc/group` ou `/etc/gshadow`.

vipw peut être utilisé pour éditer les fichiers `/etc/passwd` ou `/etc/shadow`.

libmisc...

libshadow contient des fonctions utilisées par la plupart des programmes de ce package.

Dépendances d'installation de Shadow

Shadow dépend de Bash, Binutils, Bison, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Sysklogd

Pour les instructions d'installation, voir la section intitulée *Installer Sysklogd−1.4.1* dans Chapitre 6.

Site officiel de téléchargement

Sysklogd (1.4.1) : http://www.infodrom.org/projects/sysklogd/

Contenu de Sysklogd

Le package Sysklogd contient des programmes pour enregistrer les messages de trace du système, tels que ceux donnés par le noyau.

Programmes installés : klogd et syslogd

Descriptions courtes

klogd est un démon système interceptant et traçant les messages du noyau.

syslogd trace les messages que les programmes système proposent. Chaque message tracé contient au moins une date et un nom d'hôte, ainsi que le nom du programme, mais cela dépend de la confiance donnée au

démon de traces.

Dépendances d'installation de Sysklogd

Sysklogd dépend de Binutils, Coreutils, GCC, Glibc, Make.

Sysvinit

Pour les instructions d'installation, voir la section intitulée *Installer Sysvinit−2.85* dans Chapitre 6.

Site officiel de téléchargement

Sysvinit (2.85): ftp://ftp.cistron.nl/pub/people/miquels/sysvinit/

Contenu de Sysvinit

Le package Sysvinit contient des programmes pour contrôler le démarrage, l'exécution et l'arrêt de tous les autres programmes.

Programmes installés : halt, init, killall5, last, lastb (lien vers last), mesg, pidof (lien vers killall5), poweroff (lien vers halt), reboot (lien vers halt), runlevel, shutdown, sulogin, telinit (lien vers init), utmpdump et wall

Descriptions

halt appelle normalement shutdown avec l'option the −h, sauf lorsqu'il se trouve déjà au niveau d'exécution 0, puis il indique au noyau d'arrêter le système. Mais il note tout d'abord dans le fichier `/var/log/wtmp` que le système est en cours d'arrêt.

init est la mère de tous les processus. Il lit ses commandes à partir de `/etc/inittab`, qui lui indique habituellement quels scripts lancés pour tel niveau d'exécution, ainsi que le nombre de gettys à lancer.

killall5 envoie un signal à tous les processus, sauf les processus de sa propre session de façon à ne pas tuer le shell qui l'a appelé.

last affiche les derniers utilisateurs à s'être connecté (et déconnecté), grâce au fichier `/var/log/wtmp`. Il peut aussi afficher les démarrages et arrêts du système, ainsi que les changements de niveaux d'exécution.

lastb affiche les tentatives échouées de connexion à partir des traces contenues `/var/log/btmp`.

mesg contrôle quels autres utilisateurs peuvent envoyer des messages vers le terminal de l'utilisateur actuel.

pidof rapporte les PID des programmes indiqués.

poweroff indique au noyau d'arrêter le système et de couper le système. Mais, voir plutôt halt.

reboot indique au noyau de redémarrer le système. Mais, voir plutôt halt.

runlevel rapporte le niveau d'exécution précédent et actuel, comme indiqué dans l'enregistrement des niveaux d'exécution, `/var/run/utmp`.

shutdown arrête le système d'une façon sécurisée, en le signalant à tous les processus et en notifiant tous les utilisateurs connectés.

sulogin permet au superutilisateur de se connecter. Il est normalement appelé par init quand le système passe en mode simple utilisateur.

telinit indique à init dans quel niveau d'exécution aller.

utmpdump affiche le contenu du fichier de connexion donné dans un format lisible facilement.

wall écrit un message à tous les utilisateurs connectés.

Dépendances d'installation de Sysvinit

Sysvinit dépend de Binutils, Coreutils, GCC, Glibc, Make.

Tar

Pour les instructions d'installation, voir la section intitulée *Installer Tar−1.13.25* dans Chapitre 6.

Site officiel de téléchargement

Tar (1.13.25) : ftp://alpha.gnu.org/gnu/tar/

Contenu de Tar

Tar est un programme d'archivage permettant de stocker et d'extraire des fichiers à partir d'un fichier d'archive connu comme un fichier tar.

Programmes installés : rmt et tar

Descriptions courtes

rmt est utilisé pour manipuler à distance un lecteur de cartouches, au travers d'une communication inter−processus.

tar est utilisé pour créer des archives tar et extraire des fichiers de celles−ci.

Dépendances d'installation de Tar

Tar dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Sed.

Tcl

Pour les instructions d'installation, voir la section intitulée *Installer Tcl−8.4.4* dans Chapitre 5.

Site officiel de téléchargement

Tcl (8.4.4): http://download.sourceforge.net/tcl/ ftp://download.sourceforge.net/pub/sourceforge/tcl/

Contenu de Tcl

(Dernière vérification effectuée auprès de la version 8.4.4.)

Le package Tcl contient le langage de commandes outils (Tool Command Language).

Tcl installe les fichiers suivants:

Programmes installés : tclsh (lien vers tclsh8.4), tclsh8.4

Bibliothèque installée : libtcl8.4.so

Descriptions courtes

tclsh8.4 est le shell de commandes Tcl.

libtcl8.4.so est la bibliothèque Tcl.

Dépendances d'installation de Tcl

Tcl dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Sed.

Texinfo

Pour les instructions d'installation, voir la section intitulée *Installer Texinfo−4.6* dans Chapitre 6.

Site officiel de téléchargement

Texinfo (4.6): ftp://ftp.gnu.org/gnu/texinfo/

Contenu de Texinfo

Le package Texinfo contient des programmes utilisés pour lire, écrire et convertir des documents Info, qui apporte une documentation système.

Programmes installés : info, infokey, install−info, makeinfo, texi2dvi et texindex

Descriptions courtes

info est utilisé pour lire les documents Info. Ces documents sont un peu comme des pages man, mais souvent vont plus en profondeur qu'une simple explication des options. Comparez par exemple man tar et info tar.

infokey compile un fichier source contenant des personnalisations Info en un format binaire.

install−info est utilisé pour installer des fichiers Info. Il met à jour les entrées du fichier index Info.

makeinfo traduit les documents Texinfo en différents formats: Info, texte ou HTML.

texi2dvi est utilisé pour formatter le document Texinfo donné en un fichier indépendant du périphérique et pouvant être imprimé.

texindex est utilisé pour trier des fichiers index Texinfo.

Dépendances d'installation de Texinfo

Texinfo dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed.

Util−linux

Pour les instructions d'installation, voir la section intitulée *Installer Util−linux−2.12* dans Chapitre 6.

Site officiel de téléchargement

Util−linux (2.12): http://ftp.cwi.nl/aeb/util−linux/

Contenu de Util−linux

Le package Util−linux contient un certain nombre d'utilitaires divers. Certains des plus connus d'entre eux sont utilisés pour monter, démonter, formatter, partionner et gérer des disques durs, ouvrir des ports tty ainsi que récupérer des messages du noyau.

Programmes installés : agetty, arch, blockdev, cal, cfdisk, chkdupexe, col, colcrt, colrm, column, ctrlaltdel, cytune, ddate, dmesg, elvtune, fdformat, fdisk, fsck.cramfs, fsck.minix, getopt, hexdump, hwclock, ipcrm, ipcs, isosize, kill, line, logger, look, losetup, mcookie, mkfs, mkfs.bfs, mkfs.cramfs, mkfs.minix, mkswap, more, mount, namei, parse.bash, parse.tcsh, pg, pivot_root, ramsize (lien vers rdev), raw, rdev, readprofile, rename, renice, rev, rootflags (lien vers rdev), script, setfdprm, setsid, setterm, sfdisk, swapoff (lien vers swapon), swapon, test.bash, test.tcsh, tunelp, ul, umount, vidmode (lien vers rdev), whereis et write

Descriptions courtes

agetty ouvre un port tty, demande un nom de connexion, puis invoque le programme login.

arch indique l'architecture de la machine.

blockdev vous permet d'appeller les ioctl de périphériques blocs à partir de la ligne de commande.

cal affiche un calendrier simple.

cfdisk est utilisé pour manipuler la table des partitions du périphérique donné.

chkdupexe trouve les exécutables dupliqués.

col filtre les retours chariots inversés.

colcrt est utilisé pour filtrer la sortie de nroff pour les terminaux manquant de capacités tels que le surlignage et les demi−lignes.

colrm filtre les colonnes données.

column formatte un fichier donné en de multiples colonnes.

ctrlaltdel initialise la fonction de la combinaison Ctrl+Alt+Del pour effectuer une réinitialisation logiciel ou matériel.

cytune était utilisé pour paramètrer finement les paramètres des pilotes de lignes série pour les cartes Cyclades.

ddate donne la date Discordien, ou convertit la date du calendrier Grégorien en Discordien.

dmesg affiche les messages du démarrage du noyau.

elvtune peut être utilisé pour paramètrer finement les performances et la réactivité d'un périphérique bloc.

fdformat réalise un formattage bas niveau d'une disquette.

fdisk pourrait être utilisé pour manipuler la table de partitions du périphérique donné.

fsck.cramfs réalise une vérification du système de fichiers Cramfs sur le périphérique indiqué.

fsck.minix réalise une vérification du système de fichiers Minix sur le périphérique indiqué.

getopt analyse les options sur la ligne de commande.

hexdump affiche le fichier indiqué en hexadécimal ou dans tout autre format indiqué.

hwclock est utilisé pour lire ou initialiser l'horlogoe matériel du système (aussi appelée horloge RTC ou BIOS).

ipcrm supprime les ressources IPC indiquées.

ipcs donne les informations de statut sur IPC.

isosize indique la taille d'un système de fichiers iso9660.

line copie une seule ligne.

logger copie le message indiqué dans les traces du système.

look affiche les lignes commençant avec la chaîne indiquée.

losetup est utiliser pour paramètrer et contrôler les périphériques loop.

mcookie génère un "magic cookie", des nombres sur 128 bits au hasard, pour xauth.

mkfs est utilisé pour construire un système de fichiers sur un périphérique (habituellement une partition du disque dur).

mkfs.bfs crée un système de fichiers bfs SCO.

mkfs.cramfs crée un système de fichiers cramfs.

mkfs.minix crée un système de fichiers Minix.

mkswap initialise le périphérique ou le fichier donné pour être utilisé comme partition de swap.

more est un filtre pour afficher du texte écran par écran. Mais less est bien meilleur.

mount attache le système de fichiers du périphérique donné sur le répertoire indiqué.

namei affiche les liens symboliques dans les chemins donnés.

pg affiche un fichier texte un écran à la fois.

pivot_root fait que le système de fichiers indiqué soit la nouvelle racine pour le processus actuel.

ramsize pourrait être utilisé pour initialiser la taille d'un disque RAM dans une image démarrable.

rdev pourrait être utilisé pour requêter et initialiser le périphérique racine et d'autres choses sur une image démarrable.

readprofile lit l'information de profilage du noyau.

rename renomme les fichiers indiqués, en remplaçant une chaîne donnée avec une autre.

renice est utilisé pour altérer la priorité des processus en cours.

rev inverse les lignes d'un fichier donné.

rootflags pourrait être utilisé pour initialiser les options de la racine dans une image démarrable.

script crée un script de la session du terminal, de tout ce qui est affiché à l'écran.

setfdprm initialise les paramètres d'une disquette apportée par l'utilisateur.

setsid lance le programme indiqué dans une nouvelle session.

setterm est utilisé pour initialiser les attributs du terminal.

sfdisk est un manipulateur de table des partitions d'un disque.

swapdev pourrait être utilisé pour initialiser le périphérique swap d'une image démarrable.

swapoff désactive la gestion des pages et du swap pour les périphériques et fichiers.

swapon active la gestion des pages et du swap pour les périphériques et fichiers.

tunelp est utilisé pour paramètrer finement la configuration d'une imprimante matricielle.

ul est un filtre pour transformer les tirets bas en séquences d'échappement de soulignement pour le terminal en cours d'utilisation.

umount déconnecte un système de fichiers à partir du répertoire racine.

vidmode pourrait être utilisé pour initialiser le mode vidéo dans une image démarrable.

whereis indique l'emplacement du binaire, des sources et de la page man pour la commande donnée.

write envoie un message à l'utilisateur indiqué sauf si l'utilisateur a désactivé de tels messages.

Dépendances d'installation de Util−linux

Util−linux dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Gettext, Glibc, Grep, Make, Ncurses, Sed, Zlib.

Vim

Pour les instructions d'installation, voir la section intitulée *Installer Vim−6.2* dans Chapitre 6.

Site officiel de téléchargement

Vim (6.2): ftp://ftp.vim.org/pub/editors/vim/unix/

Contenu de Vim

Le package Vim contient un éditeur de texte configurable construit pour permettre une édition efficace du texte.

Programmes installés : efm_filter.pl, efm_perl.pl, ex (lien vers vim), less.sh, mve.awk, pltags.pl, ref, rview (lien vers vim), rvim (lien vers vim), shtags.pl, tcltags, vi (lien vers vim), view (lien vers vim), vim, vim132, vim2html.pl, vimdiff (lien vers vim), vimm, vimspell.sh, vimtutor et xxd

Descriptions courtes

efm_filter.pl est un filtre pour créer un fichier d'erreur pouvant être lu par vim.

efm_perl.pl reformatte les messages d'erreur de l'interpréteur Perl pour être utilisé avec le mode quickfix de vim.

ex lance vim en mode ex.

less.sh est un script lançant vim avec less.vim.

mve.awk s'occupe des erreurs de vim.

pltags.pl crée un fichier de balises pour du code perl, utilisé par vim.

ref vérifie l'orthographe des arguments.

rview est une version restreinte de view: aucune commande shell ne peut être lancée et view ne peut être suspendu.

rvim est une version retreinte de vim: aucune commande shell ne peut être lancée et vim ne peut être suspendu.

shtags.pl génère un fichier de balises pour des scripts perl.

tcltags génère un fichier de balises pour du code TCL.

vi lance vim en mode compatible vi.

view lance vim en mode lecture−seule.

vim est l'éditeur.

vim132 lance vim avec un terminal en mode 132 colonnes.

vim2html.pl convertit la documentation de vim en HTML.

vimdiff édite deux ou trois version d'un fichier avec vim et affiche les différences.

vimm active le modèle d'entrée DEC sur un terminal distant.

vimspell.sh est un script qui épelle un fichier et génère les instructions de syntaxe nécessaire pour la coloration dans vim.

vimtutor vous apprend les touches et commandes de base de vim.

xxd crée un vidage mémoire d'un fichier donné. Il peut aussi faire le contraire, de façon à pouvoir être utilisé pour corriger des binaires.

Dépendances d'installation de Vim

Vim dépend de Bash, Binutils, Coreutils, Diffutils, GCC, Glibc, Grep, Make, Ncurses, Sed.

Zlib

Pour les instructions d'installation, voir la section intitulée *Installer Zlib−1.1.4* dans Chapitre 6.

Site officiel de téléchargement

Zlib (1.1.4) : http://www.gzip.org/zlib/ Correctif "Zlib Vsnprintf" : http://www.linuxfromscratch.org/patches/lfs/5.0/zlib−1.1.4−vsnprintf.patch

Contenu de Zlib

Le package Zlib contient la bibliothèque zlib, utilisé par certains programmes pour ses fonctions de compression et de décompression.

Bibliothèques installées : libz[a,so]

Descriptions

libz* contient des fonctions de compression et décompression utilisées par certains programmes.

Dépendances d'installation de Zlib

Zlib dépend de Binutils, Coreutils, GCC, Glibc, Make, Sed.

Annexe B. Index des programmes et des packages

Voici une liste de tous les programmes et de toutes les bibliothèques installés dans ce livre, avec pour chacun un lien vers le package, auquel il appartient, en Annexe A.

- a2p : Perl
- acinstall : Automake
- aclocal : Automake
- addftinfo : Groff
- addr2line : Binutils
- afmtodit : Groff
- agetty : Util−linux
- apropos : Man
- ar : Binutils
- arch : Util−linux
- arp : Net−tools
- as : Binutils
- attrs : Perl
- autoconf : Autoconf
- autoheader : Autoconf
- autom4te : Autoconf
- automake : Automake
- autopoint : Gettext
- autoreconf : Autoconf
- autoscan : Autoconf
- autoupdate : Autoconf
- awk : Gawk
- badblocks : E2fsprogs
- basename : Coreutils
- bash : Bash
- bashbug : Bash
- bigram : Findutils
- bison : Bison
- blkid : E2fsprogs
- blockdev : Util−linux
- bunzip2 : Bzip2
- bzcat : Bzip2
- bzcmp : Bzip2
- bzdiff : Bzip2
- bzegrep : Bzip2
- bzfgrep : Bzip2
- bzgrep : Bzip2
- bzip2 : Bzip2
- bzip2recover : Bzip2
- bzless : Bzip2
- bzmore : Bzip2
- c++ : GCC
- c++filt : Binutils
- c2ph : Perl
- cal : Util−linux