

fli4l – Documentation pour développeur

Version 4.0.0-trunk-x86-r60730

Frank Meyer
courriel: frank@fli4l.de

L'équipe fli4l
courriel: team@fli4l.de

29 août 2022

Table des matières

1	Documentation pour développeur	4
1.1	Règles générales	4
1.2	Compiler les programmes	4
1.3	Concept modulaire	5
1.3.1	mkfli4l	5
1.3.2	Structure	5
1.3.3	Configuration du paquetage	5
1.3.4	Liste des fichiers à copier	7
1.3.5	Analyse des variables de configuration	11
1.3.6	Définitions pour contrôler les variables de configuration	13
1.3.7	Contrôle détaillé de la configuration	18
1.3.8	Supporte différent choix de version du Kernel	33
1.3.9	Documentation	33
1.3.10	Formats de fichier	35
1.3.11	Développer la documentation	35
1.3.12	Programme-Client	35
1.3.13	Code source	35
1.3.14	Les autres fichiers	36
1.4	Conditions générales de création de script pour fli4l	36
1.4.1	Structure	36
1.4.2	Gestion des variables de configuration	37
1.4.3	Recherche d'erreur	38
1.4.4	Remarques	39
1.5	Utiliser le filtrage de paquets	39
1.5.1	Ajouter vos propres chaînes et règles	39
1.5.2	Classer les règles dans une infrastructure	40
1.5.3	Extension pour le test de filtrage de paquets	41
1.6	Création d'un CGI pour le paquetage <i>httpd</i>	42
1.6.1	Informations générales sur le serveur Web	42
1.6.2	Nom du script	43
1.6.3	Configuration du menu	43
1.6.4	Construction d'un script CGI	44
1.6.5	Divers	49
1.6.6	Dépannage	49
1.7	Démarrer, arrêter, se connecter et se déconnecter avec fli4l	50
1.7.1	Concept de Boot	50
1.7.2	Scripts de démarrage et d'arrêt	50
1.7.3	Fonctions auxiliaires	52
1.7.4	Règles mdev	54
1.7.5	Périphériques ttyI	57

Table des matières

1.7.6	Scripts de connexion et de déconnexion par modem	58
1.8	Paquetage Template	59
1.9	Construction du Boot sur un support de données	59
1.10	Fichiers de configurations	60
1.10.1	Configuration du fournisseur	60
1.10.2	Configuration DNS	61
1.10.3	Fichier hôte	61
1.10.4	Configuration de imond	61
1.10.5	Le fichier <code>/etc/.profile</code>	62
1.10.6	Les scripts dans <code>/etc/profile.d/</code>	62
Table des figures		63
Liste des tableaux		64
Index		65

1 Documentation pour développeur

1.1 Règles générales

Certaines règles doivent être respectées, lorsque vous ajoutez un nouveau paquetage dans la base de données OPT-fli4l, qui se trouve sur la page d'accueil du site fli4l. Le paquetage qui ne respectent pas à ces règles, sera supprimé de la base de données, sans avertissement préalable.

1. L'utilisateur ne doit faire, AUCUNE copie supplémentaire! fli4l fournit un système sophistiqué, Les données des paquetages-fli4l sont décompressés dans les répertoires de l'installation, tous les fichiers qui font fonctionner le routeur sont dans le répertoire `opt/`.
2. Les paquetages correctement emballer sont compressés : de telle sorte, que les paquetages soient facilement décompresser dans le répertoire-fli4l.
3. Les paquetages doivent être TOTALEMENT configurable dans le fichier de configuration. L'utilisateur ne doit pas faire de modifications sur d'autres fichiers de configurations. Vous ne devez pas mettre l'utilisateur en difficulté, sur des décisions difficiles à prendre, par exemple (à la fin du fichier de configuration avec une remarque en gros caractère : ONLY MODIFY IF YOU KNOW WHAT YOU DO).
4. Encore une remarque sur le fichier de configuration : les nom des variables doivent être claires et l'on doit savoir à quelle OPT elles appartiennent, par exemple `OPT_HTTPD` les nom des variables sont `OPT_HTTPD`, `HTTPD_USER_N`, etc.
5. S'il vous plaît, si vous avez compilé vous-même de petits (Programmes) binaire! Et si vous traduisez vous-même le FBR, pensez de désactiver les fonctionnalités inutiles.
6. Contrôler votre Copyright! Si vous utilisé un modèle de fichier, merci de respecter les droit d'auteur. Le copyright doit est remplacé ici par votre propre nom si vous créez vos fichiers. vérifié en particulier, les fichiers dans `config-`, `Check-` et les fichiers textes dans `opt-`. Si vous copiez la documentation mot à mot, le copyright de l'auteur d'origine doit être naturellement gardé!
7. Merci de diffuser seulement des types d'archivages, utilisant des formats libres. Il s'agit notamment de :
 - ZIP (`.zip`)
 - GZIP (`.tgz` ou `.tar.gz`)S'il vous plaît n'utilisez pas les autres formats tels que RAR, ACE, Blackhole, LHA, etc. Vous ne devez pas utiliser les fichiers d'installation Windows (`.msi`) ou les fichiers d'installation (`.exe`) et les archives auto-extractable.

1.2 Compiler les programmes

Pour pouvoir compiler des programmes vous allez avoir besoin du paquetage « src » qui est disponible à part. Il y a également une documentation pour compiler vos propre programme pour fli4l.

1.3 Concept modulaire

Depuis la version 2.0, fli4l est réparti en modules (ou paquetages), par exemple

- fli4l-4.0.0-trunk-x86-r60730 <— paquetage base
- dns-dhcp
- dsl
- isdn
- sshd
- et bien d'autres ...

Avec le paquetage base, fli4l est un simple routeur Ethernet. Pour ISDN et/ou DSL, vous devez décompacter le paquetage ISDN et/ou DSL dans le répertoire fli4l. Il en va de même pour les autres paquetages.

1.3.1 mkfli4l

À partir des paquetages et en fonction des paramètres spécifiques des fichiers de configurations, les fichiers suivants seront créés, le fichier avec les configurations, `rc.cfg` et deux archives `rootfs.img` et `opt.img`, ces fichiers sont produits à l'aide du programme `mkfli4l`, celui-ci lie les différents paquetages et contrôle les fichiers de configurations pour repérer d'éventuelles erreurs.

`mkfli4l` accepte les options indiqués dans le tableau 1.1. Si aucune valeur n'est indiquée, les valeurs par défaut entre les parenthèses seront prises en compte. Vous pouvez voir la liste complète des options dans (le tableau 1.1) avec la commande :

```
mkfli4l -h
```

1.3.2 Structure

Un paquetage peut contenir plusieurs opts, mais si en général le paquetage contient seulement un opt, il est opportun de nommer le paquetage du même nom que l'OPT. Vous pourrez ensuite remplacer <PAQUETAGE> par le nom des paquetages respectif. Un paquetage comprend les éléments suivants :

- Gestionnaire de fichiers
- Documentation
- Documentation du développeur
- Programme client
- Code source
- Autres fichiers

Les différentes parties sont détaillées ci-dessous.

1.3.3 Configuration du paquetage

Dans le fichier `config/<PAQUETAGE>.txt`, les modifications de la configuration du paquetage sont réalisées par l'utilisateur. Toutes les variables du fichier de configuration doivent commencer uniformément par le nom de ou des OPTs, par exemple :

```
#-----  
# Optional package: TELNETD
```

TABLE 1.1 – Paramètres pour mkfli41

Option	Signification
-c, --config	Avec cette option, on définit le répertoire dans lequel mkfli41 recherche les fichiers de config des paquetages ; (par défaut : config)
-x, --check	On place ici la liste des fichiers des paquets, dans le quel mkfli41 doit contrôler et rechercher les erreurs (<PAQUETAGE>.txt, <PAQUETAGE>.exp et <PAQUETAGE>.ext ; (par défaut : check)
-l, --log	On place ici le fichier journal, dans le quel mkfli41 enregistre les messages d'erreurs et les avertissements ; (par défaut : img/mkfli41.log)
-p, --package	On place ici le paquetage qui doit être examiné, on peut indiquer dans cette option plusieurs paquetages à la fois pour qu'ils soient examinés. Cependant si vous utilisez l'option -p, l'option <check_dir>/base.exp est en principe d'abord vérifiée, afin de gérer les paramètres général du paquetage de base.
-i, --info	Avec cette option, on affiche des informations sur le déroulement de la construction (lecture des fichiers, vérification des fichiers, les problèmes rencontrés seront affichés lors du processus de contrôle).
-v, --verbose	Même signification que la variante -i
-h, --help	Avec cette option, on affiche l'aide
-d, --debug	Avec cette option, vous pouvez déboguer le processus de construction. Cette option sert surtout pour les développeurs de paquetage, qui souhaitent en savoir un peu plus, sur le déroulement du contrôle des paquetages.
Debugoption	Signification
check	show check process
zip-list	show generation of zip list
zip-list-skipped	show skipped files
zip-list-regexp	show regular expressions for zip list
opt-files	check all files in opt/<PAQUETAGE>.txt
ext-trace	show trace of extended checks

```
#-----
OPT_TELNETD='no'          # install telnetd: yes or no
TELNETD_PORT='23'        # telnet port, see also FIREWALL_DENY_PORT_x
```

Le fichier de configuration de l'OPT doit être configuré en conséquence avec une en-tête (voir ci-dessus). Cela augmente la clarté, en particulier si le paquetage contient plusieurs OPTs. Les variables associées à l'OPT - ne doivent pas être écrites en retrait - également pour plus de clarté. Les commentaires et lignes vides sont autorisés, les commentaires doivent commencer de manière uniforme jusqu'à la 33ème colonne. Si une variable ou un commentaire occupe plus de 32 caractères, la ligne sera décalée à la 33ème colonne. Les commentaires plus longs doivent être séparés dès la 33ème colonne pour commencer une nouvelle ligne. Cette mesure vise à améliorer la lisibilité du fichier de configuration.

Toutes les valeurs derrière le signe égale doivent être écrites entre des guillemets¹, autrement, lors du boot vous pouvez avoir des problèmes.

Les variables incluses (voir plus bas) dans le fichier `rc.cfg` sont activées, toutes les autres sont ignorées. La seule exception est la variable qui porte le nom `<PAQUETAGE>_DO_DEBUG`. Celle-ci sert pour déboguer des paquetages et est enregistrée en globalité.

1.3.4 Liste des fichiers à copier

Le fichier `opt/<PAQUETAGE>.txt` doit contenir les instructions suivantes.

- Quels fichiers correspond, à quel OPT,
- Quand le fichier `opt.img` ou `rootfs.img` doit être transféré et généré,
- Quel ID-utilisateur (uid), ID-groupe (gid) et les droits qu'ils doivent obtenir,
- Quel conversion doit être faite à l'archive avant l'enregistrement.

Quand `mkfli41` est exécuté il se base sur les archives nécessaires à l'installation.

Les lignes vides et commençant par « # » sont ignorées. Dans l'une des deux premières lignes, vous devez indiquer la version du format du paquetage, qui doit être :

<première colonne>	<deuxième colonne>	<troisième colonne>
<code>opt_format_version</code>	1	-

Les autres lignes ont la syntaxe suivante :

<première colonne>	<deuxième colonne>	<troisième colonne>	<colonne suivante>
Variable	valeur	Fichier	Option

1. Dans la première colonne se trouve les noms des variables, elle dépend du fichier appliqué de la troisième colonne. Si le nom de la variable apparaît plusieurs fois dans la première colonne, cela veut dire que plusieurs fichiers en dépendent. Chaque variable qui est placée dans le fichier `opt/<PAQUETAGE>.txt`, est marqué par `mkfli41`.

Si plusieurs variables doivent être testés avec la même valeur, une liste de variables peut être utilisée (elles seront séparés par une virgule). Dans ce cas, il sera suffisant d'enregistrer au moins *une* valeur requise dans la deuxième colonne pour toutes les variables. Il est important de ne *pas* mettre d'espace entre les variables !

1. Il est possible d'utiliser les guillemets simples ou les guillemets doubles, on peut donc écrire `F00='bar'` ou `F00="bar"`. Cependant, l'utilisation des guillemets doubles doivent être exceptionnelle, vous devez vérifier absolument comment le shell-Unix traite des guillemets simples et doubles

Pour les variables OPT (se sont les variables qui commence par OPT_ et qui accepte les valeurs YESNO) le préfixe ogOPT_fg peut être omis. En outre, il n'y a pas d'importance si les variables sont indiquées en majuscule, en minuscule (ou mixte).

2. Dans la deuxième colonne sont indiquées les valeurs. Si la variable de la première colonne correspond à la valeur indiqué dans la deuxième colonne et si elle est activé, (voir plus bas), de plus si vous avez dans la première colonne %-Variable qui se répète pour chaque indice différent, alors ces variables vérifient la valeur de la deuxième colonne, si elle correspond, alors tous les fichiers de la troisième colonne seront copiés. Il est à noter qu'en raison des valeurs des même variables un seul fichier sera copié.

Il est possible d'indiquer le caractère og!fg devant la valeur. Dans ce cas, le test est annulé, ce qui signifie que le fichier sera copié seulement si la variable ne contient *pas* de valeur.

3. Dans la troisième colonne est indiqué le nom du fichier. Le chemin est relatif au répertoire **opt**. Le fichier doit exister et être lisible, sinon, il y aura une erreur et le programme **mkfli41** qui génère automatiquement la construction du média de boot s'arrêtera.

Si le nom du fichier commence par **rootfs:-**préfixe, le fichier sera copié dans la liste et sera inclus dans le RootFS avec les autres fichiers. Si il y a un préfixe au fichier il sera supprimé avant la copie.

Si le fichier se trouve dans le sous répertoire config, il sera ajouté dans la liste des fichiers du répertoire config, mais ce fichier ne pourra pas avoir de préfixe **rootfs:-**, comme les fichiers qui proviennent du sous répertoire **opt**.

Si le fichier à copier est un module-kernel, on peut remplacer le nom de la version du kernel par `#{KERNEL_VERSION}`. **mkfli41** prendra alors la version configurée et l'intégrera. Cela permet d'avoir un packaging de module-kernel pour chaque versions différente, en plus la bonne version du kernel sera toujours copiée sur le routeur. Au sujet des modules-kernel le chemin peut être complètement omis, car **mkfli41** trouve le chemin des modules en utilisant les fichiers **modules.dep** et **modules.alias**, reportez-vous à la section « [Résolution automatique pour la traçabilité des modules-kernel](#) » (Page 10)

4. Vous pouvez voir dans la colonne du tableau 1.2 le détail des options sur les propriétés, les groupes, les droits d'accès des fichiers et les conversions.

Quelques exemples :

- Copie le fichier, si OPT_TELNETD='yes', placez uid/gid pour le root et placez les droits sur 755 (rwxr-xr-x).

```
telnetd    yes    usr/sbin/in.telnetd mode=755
```
- Copie le fichier, placez uid/gid pour le root, les droits sur 555 (r-xr-xr-x) et converti le fichier dans le format Unix en même temps il supprime tous les caractères superflus.

```
base      yes    etc/rc0.d/rc500.killall mode=555 flags=sh
```
- Copie le fichier, si PCMCIA_PCIC='i82365', placez uid/gid pour le root et placez les droits sur 644 (rw-r--r)

```
pcmcia_pcic i82365 lib/modules/#{KERNEL_VERSION}/pcmcia/i82365.ko
```
- Copie le fichier, si l'une des variables NET_DRV_% correspond à la deuxième colonne, placez uid/gid pour le root et placez les droits sur 644 (rw-r--r)

```
net_drv_%   3c503  3c503.ko
```
- Copie le fichier, si la variable POWERMANAGEMENT ne contient *pas* la valeur « none » :

```
powermanagement !none etc/rc.d/rc100.pm mode=555 flags=sh
```


TABLE 1.2 – Options pour les fichiers

Option	Signification	Valeur standard
type=	type d'entrée : <i>local</i> Objet fichiers-système <i>file</i> Fichier <i>dir</i> Répertoire <i>node</i> Matériel <i>symlink</i> Lien (symbolique) Si elle est présente, cette option doit venir en premier. Le type « local » représente un type d'objet existant dans le fichier système et correspond donc (le cas échéant) à « file », « dir », « node » ou « symlink ». Les autres types, à l'exception de « file » peuvent être utilisés pour enregistrer des archives, il ne doivent pas être présent dans le fichier système local. Par ex., vous pouvez les utiliser pour créer des fichiers de périphérique dans l'archive-RootFS.	local
uid=	Propriété du fichier, soit numériquement, soit en tant que mot de passe	root
gid=	Groupe du fichier, soit numériquement, soit en tant que nom de groupe	root
mode=	Pour les droits d'accès	Les Fichiers et matériels : rw-r--r-- (644) Les Répertoires : rw-r--r-- (755) Les liens : lrwxrwxrwx (777)
Les flags= (type=file)	Conversion avant l'enregistrement dans l'archive : <i>utxt</i> Conversion au format-Unix <i>dtxt</i> Conversion au format-DOS <i>sh</i> Script Shell : Conversion dans le format-UNIX, suppression des signes inutiles <i>luac</i> Script Lua : Traduction en bytecode de Lua-VM	
name=	Nom alternatif sous lequel l'entrée est enregistrée dans l'archive	
devtype= (type=node)	Décrit le type de matériel (« c » pour s'orienter vers la marque et « b » pour s'orienter vers le matériel de bloc. Doit être à la deuxième place.	
major= (type=node)	Décrit le nombre soi-disant « Majeur » -Numéro de fichier du périphérique. Doit être à la troisième place.	
minor= (type=node)	Décrit le nombre soi-disant « Mineur » -Numéro de fichier du périphérique. Doit être à la quatrième place.	
linktarget= (type=symlink)	Décrit la cible du lien symbolique. Doit être à la deuxième place.	

- Copie le fichier, si l'une des variables OPT OPT_MYOPTA ou OPT_MYOPTB contient la valeur « yes » :

```
myopta,myoptb yes usr/local/bin/myopt-common.sh mode=555 flags=sh
```

Cet exemple est juste un raccourci pour :

```
myopta yes usr/local/bin/myopt-common.sh mode=555 flags=sh
myoptb yes usr/local/bin/myopt-common.sh mode=555 flags=sh
```

Et celui-ci est un raccourci pour :

```
opt_myopta yes usr/local/bin/myopt-common.sh mode=555 flags=sh
opt_myoptb yes usr/local/bin/myopt-common.sh mode=555 flags=sh
```

- Copie du fichier opt/usr/bin/beep.sh dans l'archive rootfs, mais sera renommer bin/beep avant :

```
base yes rootfs:usr/bin/beep.sh mode=555 flags=sh name=bin/beep
```

Les fichiers sont uniquement copiés, si les conditions indiquées plus haut sont remplies, si la variable est placée sur `OPT_PAQUETAGE='yes'`. Quelle variable-OPT est associée ? Pour la vérification voir le fichier `check/<PAQUETAGE>.txt`

Si une variable est référencée dans le paquetage et que celle-ci n'est pas définie, il peut arriver que le paquetage correspondant n'est pas installé. Cela mènerait à un message d'erreur du programme `mkfli41`, car `mkfli41` attend que toutes les variables référencées dans le fichier `opt/<PAQUETAGE>.txt` soient définies.

Pour gérer correctement cette situation, on introduit la fonction-« weak » dans le format suivant :

```
weak      variable  -
```

Si la variable est définie, même si elle n'est pas disponible la valeur de celle-ci sera indiquée « undefiniert ». Toutefois, il convient de noter ici que le préfixe « `OPT_` » ne doit *pas* être omis (s'il existe), sinon la variable sera définie *sans* ce préfixe.

Un exemple du fichier `opt/rrdtool.txt` :

```
weak opt_openvpn -
[...]
openvpn  yes      usr/lib/collectd/openvpn.so
```

Sans la définition `weak`, la commande `mkfli41` affichera un message d'erreur si vous utilisez le paquetage « `rrdtool` » et si le paquetage « `openvpn` » n'est pas aussi présent. La définition `weak` est également utilisé dans le cas où le paquetage « `openvpn` » n'existe pas, il n'y aura aucun message d'erreur.

Fichiers de configurations spécifiques

Dans certaine situations, on aimerait remplacer des fichiers de configurations originaux par des fichiers spécifiques, qui serait compacté dans l'archive `opt.img`, par exemple, ajouter une Key-hôte ou ajouter son propre Scripte-Firewall, ... `varmkfli41` supporte ce scénario, il vérifie si un fichier est disponible dans le répertoire config, dans ce cas, ce fichier sera ajouté dans la liste des fichiers de l'archive `opt.img` ou `rootfs.img`.

Une autre façon d'ajouter des fichiers de configurations spécifiques dans l'archive, est décrit dans le chapitre [Contrôle de la configuration avancée](#) (Page 27).

Résolution automatique pour la traçabilité des modules-kernel

Dans certaine circonstance un module-kernel a parfois besoin d'autres modules-kernel. Ces modules doivent être chargés avant et seront également inclus dans l'archive. `mkfli41` détermine la traçabilité des modules avec les fichiers `modules.dep` et `modules.alias` (les deux fichiers sont générés à la compilation du kernel) ainsi ils ajouteront automatiquement tous les modules nécessaires dans l'archive. Par exemple l'enregistrement peut être le suivant

```
net_drv_%  ne2k-pci  ne2k-pci.ko
```

Le pilote `ne2k_pci` dépend des fichiers suivant `8390.ko`, et `crc32.ko` ils seront ajoutés en premiers dans l'archive.

L'enregistrement du `modules.dep` et du `modules.alias` dans le RootFS est nécessaire, ensuite `modprobe` utilise ces fichiers pour charger les pilotes.

1.3.5 Analyse des variables de configuration

Avec le fichier `check/<PAQUETAGE>.txt` les variables peuvent être contrôlés pour leurs validités. Ce contrôle était intégré dans les versions précédentes du programme `mkfli4l`, mais les paquetages `fli4l` sont modulaire et nous avons du installé un second-contrôle dans ce fichier. Dans ce fichier, une ligne est accessible pour chaque variable du fichier de configuration. Ces lignes se composent de quatre à cinq colonnes, qui ont les fonctions suivantes :

1. **Variable** : Cette colonne indique le nom des variables à vérifier, qui sont dans le fichier config. Cette colonne indique le nom des variables à vérifier, si c'est une *liste de variables*, elle peut apparaître plusieurs fois avec différents indices, donc à la place de l'indice le signe pourcentage (%) sera inséré dans la variable. Le signe sera toujours indiqué comme ceci « `_%` » au milieu de la variable et comme ceci « `_%` » à la fin de la variable. La variable peut contenir plusieurs signes pourcentages, de sorte à réaliser des matrices multidimensionnels. Normalement vous ne serez pas amené à voir ces variables un peut étranges avec deux signes pourcentage, tels que « `foo_%_%` ».

Dans certaine situation, nous pouvons avoir besoin de variable optionnelle supplémentaire dans le fichier config. Ces variables sont présentes dans le fichier de contrôle, elles sont marquées et précédées du signe « `+` ». Dans le tableau, vous pouvez voir aussi des variables précédées du signe « `++` ». Avec le signe « `+` », ces variables sont présentes ou absente du fichier config. Avec le signe « `++` », ces variables sont totalement absente du fichier config. Vous pouvez les rajouter manuellement, si vous en avez besoin dans le fichier config.

2. **OPT_VARIABLE** : Cette colonne indique, les variables OPT. Ces variables sont seulement vérifiées pour leurs validités, à savoir si la variable est sur « `yes` ». S'il n'y a pas de variable OPT un « `-` » sera indiqué. Dans ce cas, la variable doit être définie dans le fichier de configuration, sauf si une valeur par défaut est défini (voir ci-dessous). Le nom de la variable OPT peut être arbitraire mais doit commencer par le préfixe « `OPT_` ».

Si une variable ne dépend d'aucune variables OPT, elle sera considérée comme *active*. Si elle dépend d'une variable OPT, elle sera explicitement active, si

- la variable OPT est active et
- la variable OPT contient la valeur "yes".

Dans tous les autres cas, la variable est inactive.

Remarque : les variables OPT inactifs seront remplacées sur « `no` » par `mkfli4l` si elle sont réglées sur « `yes` » dans le fichier de configuration, un message d'avertissement sera alors généré (c'est à dire « `OPT_Y='yes' ignoré, parce que OPT_X='no'` »). Pour les chaînes de dépendance transitive (OPT_Z dépend de OPT_Y qui à son tour dépend de OPT_X) cela fonctionnera de manière fiable, si les noms de tous les variables OPT commencent par « `OPT_` ».

3. **VARIABLE_N** : Si dans la première colonne, vous avez des noms de variables avec le signe %, elles spécifient la fréquence d'apparition de la variable (c'est la variable qui s'appelle *N-variable*). Si la variable est multidimensionnel, le dernier index est en l'occurrence spécifié. Si la variable dépend d'un OPT, la N-variable doit dépendre ou pas du même OPT. Si la variable ne dépend pas d'un OPT, la N-variable ne doit également pas en dépendre. Si aucune N-variable existe, le signe « `-` » sera spécifier.

Pour la compatibilité avec les futures versions de `fli4l` la variable spécifiée ici *doit* être identique à la variable qui est dans **OPT_VARIABLE** et le dernier signe « `%` » sera remplacé

par un « N » tout ce qui suit sera retiré. Pour la liste `HOST_%_IP4` vous devez assigner N-Variable `HOST_N` et pour la liste `PF_USR_CHAIN_%_RULE_%` vous devez aussi assigner N-Variable `PF_USR_CHAIN_%_RULE_N`. *Toutes les autres définitions de N-variable ne seront pas compatibles avec les versions futures de fli4l!*

4. **VALUE** : cette colonne donne les valeurs possibles que peut prendre la variable. Vous pouvez par ex. avoir les informations suivant :

Nom	Signification
NONE	ne déclenche aucun contrôle
YESNO	La variable doit être sur « yes » ou sur « no »
NOTEMPTY	La variable ne peut pas être vide
NOBLANK	La variable ne devrait pas contenir d'espaces
NUMERIC	La variable doit être numérique
IPADDR	La variable doit être une adresse IP
DIALMODE	La variable doit être sur « on », « off » ou « auto »

Si vous indiquez le préfixe « `WARN_` », la valeur sera irrégulière et sera indiqué par un message, `mkfli4l` ne s'arrêtera pas, mais affichera seulement un avertissement.

Le contrôle est défini par des expressions régulières dans le fichier `check/base.exp`. Ce fichier à été récemment étendu, il contient maintenant des contrôles complémentaires suivants : `HEX`, `NUMHEX`, `IP_ROUTE`, `DISK` et `PARTITION`.

Si les développeurs-opt ont besoin de rajouter une entrée, le nombre de termes peut, à tout moment être étendu.

En outre, les expressions régulières peuvent être ajoutées directement dans le fichier du répertoire `check`, on peut également se référer à des expressions existantes. Par exemple au lieu d'utiliser `YESNO` on pourrait également écrire

`RE:yes|no`

cela est utile pour un test qui est effectué qu'une seule fois, il est relativement simple. Pour de plus amples informations, voir le chapitre suivant.

5. **Paramètre par défaut** : dans cette colonne, une valeur facultative par défaut pour les variables peut être définie, dans le cas où la variable n'est pas spécifié dans le fichier de configuration.

Remarque : à présent si cela ne fonctionne pas pour les variables de la liste. La variable ne doit pas être facultative, donc il ne doit pas avoir le signe « + » devant le nom de la variable.

Exemple :

`OPT_TELNETD - - YESNO "no"`

La variable `OPT_TELNETD` est maintenant manquante dans le fichier de configuration, mais dans le fichier `rc.cfg` elle sera affichée avec la valeur « no ».

La variable avec le signe pourcentage peut être mieux expliquée avec un exemple. Voici une partie du fichier `check/base.txt` :

<code>NET_DRV_N</code>	-	-	<code>NUMERIC</code>
<code>NET_DRV_%</code>	-	<code>NET_DRV_N</code>	<code>NONE</code>
<code>NET_DRV_%_OPTION</code>	-	<code>NET_DRV_N</code>	<code>NONE</code>

En d'autres termes, en fonction de la valeur indiquée dans `NET_DRV_N` les variables `NET_DRV_N`, `NET_DRV_1_OPTION`, `NET_DRV_2_OPTION`, `NET_DRV_3_OPTION`, etc. seront vérifiées.

1.3.6 Définitions pour contrôler les variables de configuration

Introduction sur les expressions régulières

Dans la version 2.0, il y a que 7 expressions, susceptibles d'examiner les variables : NONE, NOTEMPTY, NUMERIC, IPADDR, YESNO, NOBLANK, DIALMODE. Ces expressions ont été fixées dans `mkfli41` pour la vérification, ils ne sont pas extensibles et se limitent à l'essentielle aux « types de données », avec juste ce qu'il faut pour pouvoir faire le contrôle.

Dans la version 2.1 un nouveau contrôle a été créé. L'objectif de cette nouvelle création est de faire un contrôle plus flexible des variables, qui sera en mesure d'examiner des expressions plus complexes. C'est la raison pour laquelle les expressions régulières seront utilisées dans un ou plusieurs fichiers séparés. Il sera possible, d'une part, de vérifier les variables avant le contrôle par `mkfli41` et d'autre part, les développeurs pourront définir leurs propres expressions, pour la configuration et le contrôle de leur paquetage.

Vous pouvez trouver une description des expressions régulières, dans « man 7 regex », ou par exemple ici : <http://unixhelp.ed.ac.uk/CGI/man-cgi?regex+7>.

Spécification des expressions régulières

On peut spécifier des expressions de deux manières différentes :

1. Extension `exp` spécifique au paquetage, dans le fichier `check/<PAQUETAGE>.exp`

Ce fichier se trouve dans le répertoire `check` et porte le même nom que son paquetage, par ex. `base.exp`. Les expressions contiennent les définitions, qui sont référencées dans le fichier `check/<PAQUETAGE>.txt`. Ainsi, `check/base.exp` peut contrôler les définitions. Bien connu le fichier `check/isdn.exp` qui contrôle la définition de la variable `ISDN_CIRC_?_ROUTE` (à l'origine le contrôle de cette variable était absent cela a été modifié).

Chaque définition sera écrite entre deux apostrophes, la syntaxe est la suivante :

```
<Name> = '<expression régulière>' : '<le message d'erreur>'
```

Autre exemple de la `check/base.exp` :

```
NOTEMPTY = '.*[~ ]+.*'           : 'should not be empty'
YESNO     = 'yes|no'              : 'only yes or no are allowed'
NUMERIC   = '0|[1-9][0-9]*'       : 'should be numeric (decimal)'
OCTET     = '1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]'
           : 'should be a value between 0 and 255'
IPADDR    = '((RE:OCTET)\.){3}(RE:OCTET)' : 'invalid ipv4 address'
EIPADDR   = '()|(RE:IPADDR)'
           : 'should be empty or contain a valid ipv4 address'
NOBLANK   = '[^ ]+'               : 'should not contain spaces'
DIALMODE  = 'auto|manual|off'      : 'only auto, manual or off are allowed'
NETWORKS  = '(RE:NETWORK)([[:space:]]+(RE:NETWORK))*'
           : 'no valid network specification, should be one or more
             network address(es) followed by a netmask,
             for instance 192.168.6.0/24'
```

Dans les expressions régulières, vous pouvez toujours rajouter une référence, dans une définition existante. Cela est plus facile que de construire une expression régulière. Il suffit simplement d'insérer la référence de cette manière `'(RE :référence)'`. (Voir la définition de l'expression `NETWORKS` ci-dessus pour un exemple approprié.)

Les messages d'erreur ont tendance à être trop longs. Il est donc possible, de les afficher sur plusieurs lignes. Vous devez toujours utiliser au début de la ligne un espace ou une tabulation. Lors de la lecture du fichier `check/<PAQUETAGE>.exp` des espaces supplémentaires sont réduits à un seul espace et une tabulation sera remplacé par des espaces.

La configuration du fichier `check/<PAQUETAGE>.exp` pourrait alors ressembler à ce qui suit :

```
NUM_HEX          = '0x[[:xdigit:]]+'
                  : 'should be a hexadecimal number
                    (a number starting with "0x")'
```

2. Les expressions régulières sont directement installées dans le fichier de contrôle `check/<PAQUETAGE>.txt`

Certaines expressions sont utilisé qu'une seule fois, ce n'est pas la peine de définir ces expressions régulières dans le fichier `check/<PAQUETAGE>.exp`. Nous pouvons alors, enregistrer simplement ces expressions dans le fichier-Check, par ex. :

# Variable	OPT_VARIABLE	VARIABLE_N	VALUE
MOUNT_BOOT	-	-	RE:ro rw no

La variable MOUNT_BOOT ne peut qu'accepter les valeurs « ro », « rw » ou « no » toutes les autres sont rejetées

Si vous voulez référencer une expression régulière existants, vous devez simplement ajouter la référence de cette manière « (RE :...) », par ex. :

# Variable	OPT_VARIABLE	VARIABLE_N	VALUE
LOGIP_LOGDIR	OPT_LOGIP	-	RE:(RE:ABS_PATH) auto

Extension d'une expression régulière déjà existant

Si vous ajoutez un paquetage optionnel supplémentaire, vous devez ajouter une expression régulière pour que la valeur de la variable soit examinée, l'expression régulière doit être agrandie, cela se passe simplement par l'ajout une nouvelle définition des valeurs dans l'expression régulière (comme décrit plus haut). Le complément de l'expression régulière existante est copié dans le fichier `check/<PAQUETAGE>.exp`. L'expression existante sera modifiée, par le caractère « + » qui sera ajouté au premier plan. L'expression existante sera complétée par une nouvelle valeur, ainsi la nouvelle valeur est ajoutée comme l'alternative à la valeur existante. Si une autre expression utilise l'expression qui a été complétée, le complément de cette expression sera aussi valable. Vous pouvez indiquer un message d'erreur, il sera simplement ajouté après l'expression.

Voici un exemple pour les pilotes-Ethernet :

- Le paquetage-base a un ensemble de pilotes Ethernet prédéfini, on sélectionne la variable NET_DRV_x avec l'expression régulière NET_DRV pour contrôler cette variable, voici ce qui est écrit dans le fichier de contrôle :

```
NET_DRV          = '3c503|3c505|3c507|...'
                  : 'invalid ethernet driver, please choose one'
                  : ' of the drivers in config/base.txt'
```

- Nous mettons à disposition le paquetage-PCMCIA pour avoir des pilotes de périphérique supplémentaires, vous devez alors compléter la variable NET_DRV, pour avoir quelque chose comme ceci :

```
PCMCIA_NET_DRV = 'pcnet_cs|xirc2ps_cs|3c574_cs|...' : ''
+NET_DRV       = '(RE:PCMCIA_NET_DRV)' : ''
```

Maintenant, vous pouvez également sélectionner les pilotes PCMCIA supplémentaires.

Expression régulière élargie en relation avec les variables YESNO

Après avoir ajouté le pilote PCMCIA dans NET_DRV comme décrit plus haut. Si le paquetage « pcmcia » est désactivé et si vous voulez quand même choisir un pilote PCMCIA dans `config/base.txt`, sans avoir de message d'erreur à la création du support de boot. Vous pouvez modifier l'expression régulière avec la variable YESNO dans la configuration. Pour cela vous devez ajouter des parenthèses immédiatement après le nom de la variable sur l'expression pour déterminer si l'expression est étendu. Si la variable est active avec la valeur « yes », l'expression sera étendu, autrement elle ne l'est pas.

```
PCMCIA_NET_DRV      = 'pcnet_cs|xirc2ps_cs|3c574_cs|...' : ''
+NET_DRV(OPT_PCMCIA) = '(RE:PCMCIA_NET_DRV)' : ''
```

Maintenant si on veut utiliser par ex. le pilote `xirc2ps_cs` dans `config/base.txt` et si la variable est paramétré sur `OPT_PCMCIA='no'`, il y aura un message d'erreur à la création des archives.

Remarque : si cela ne fonctionne *pas*, la variable n'est peut être pas définie explicitement dans le fichier de configuration mais elle obtient une valeur avec un paramètre par défaut dans `check/<PAQUETAGE>.txt`. Dans ce cas, vous devez définir explicitement la variable et enlever le paramètre par défaut si nécessaire.

Définir une expression régulière en fonction d'autres variables

Vous pouvez alternativement utiliser toutes les valeurs de la variable, à condition, que la syntaxe apparaît comme indiquer ci-dessous :

```
+NET_DRV(KERNEL_VERSION=~'^3\.18\..*$',) = ...
```

Si l'expression `KERNEL_VERSION` correspond au donné, c'est-à-dire ici à utilisation du kernel version 3.18, alors la liste complète des pilotes réseaux qui tournent avec celui-ci seront autorisés.

Remarque : si cela ne fonctionne *pas*, la variable n'est peut être pas définie explicitement dans le fichier de configuration mais elle obtient une valeur avec un paramètre par défaut dans `check/<PAQUETAGE>.txt`. Dans ce cas, vous devez définir explicitement la variable et enlever le paramètre par défaut si nécessaire.

Message d'erreur

Si pendant le contrôle, `mkfli4l` trouve une erreur, un message de cette erreur apparaîtra exemple :

```
Error: wrong value of variable HOSTNAME: '' (may not be empty)
Error: wrong value of variable MOUNT_OPT: 'rx' (user supplied regular expression)
```

La première erreur, a été définie dans le fichier `check/<PAQUETAGE>.exp`, une indication sur l'erreur sera affichée. La deuxième erreur a été spécifié directement dans le fichier `check/<PAQUETAGE>.txt`, il n'y aura aucune indication supplémentaire sur la raison de l'erreur.

Définition des expressions régulières

Les expressions régulières sont définies comme indiqué ci-dessous :

Expression régulière : vous pouvez paramétrer une ou plusieurs options, vous devez les séparer par le signe « | », par ex. « ro|rw|no ». Si l'une des options s'applique, alors l'expression sera appliquée (ici les expressions valides sont « ro », « rw » et « no »).

Une option est une concaténation de tronçon, qui sont simplement reliés entre eux.

Un tronçon est un « Atome », suivi pas un signe quantificateur « * », « + », « ? » ou « {min, max} ». La signification est la suivante :

- « a* » - Capture le caractère a, zéro ou plusieurs fois
- « a+ » - Capture le caractère a, une ou plusieurs fois
- « a? » - Capture le caractère a, zéro ou une fois
- « a{2,5} » - Capture le caractère a, entre 2 et 5 fois
- « a{5} » - Capture le caractère a, 5 fois
- « a{2,} » - Capture le caractère a, au moins 2 fois
- « a{,5} » - un maximum de cinq « a »s

Un « atome » est

- Une expression régulière entre parenthèses, par ex. « (a|b)+ » s'applique à toute chaîne qui contient au moins un « a » ou « b », dans n'importe quel ordre
- Une paire de parenthèses vide, représente une expression « vide »
- Une expression entre les crochets « [] » (voir ci-dessous)
- Le point « . » remplace n'importe quel caractère, par ex. « .+ » s'applique à toute chaîne qui contient au moins un caractère.
- « ^ » représente le début d'une chaîne de caractères, par ex. « ^a.* » s'applique à une chaîne de caractères qui commence par un « a » et suivi par un caractère quelconque, « a » ou « adkadhashdkash ».
- « \$ » fin d'une chaîne de caractères, fin de ligne
- « \ » suivis par l'un des caractères spéciaux ^ . [\$ () | * + ? { \ correspond au caractère réel sans sa signification spéciale.
- Un caractère écrit, est exactement un caractère normale, par ex. « a » est le caractère alphabétique « a ».

Signification d'une expression entre des crochets, lire la suite.

- « x-y » - Trouvera tout les lettres qui se situe entre « x » et « y », par ex. « [0-9] » correspond à tous les caractères de 0-9, « [a-zA-Z] » correspond à toutes les lettres, qu'elles soient majuscule ou minuscule
- « ^x-y » - Trouvera n'importe quel lettres qui est en dehors des crochets par exemple « [^0-9] » s'applique à toutes les lettres mais *pas* au chiffres
- « [:character-class:] » - Trouvera une classe de caractères *character-class*. Voici les classes de caractères standard : `alnum`, `alpha`, `blank`, `digit`, `lower`, `print`, `punct`, `space`, `upper` et `xdigit`. Donc « [:alpha:] » est utilisé pour toutes les lettres majuscules et minuscules et est identique à « [:lower:][:upper:] ».

Exemples d'expressions régulières

Jetons un coup d'oeil sur quelques exemples :

NUMERIC : Valeur numérique qui est constitué d'au moins de un, mais aussi d'un nombre quelconque de chiffres. Avec le signe « + » la valeur peut être répéter au moins une ou plusieurs

fois à partir d'un nombre, voici un exemple pour obtenir un nombre composé :

```
NUMERIC = '[0-9]+'
```

ou alors

```
NUMERIC = '[:digit:]]+'
```

NOBLANK : valeur qui ne contient pas d'espace, les caractères peuvent être quelconque (à l'exception de l'espace), voici un exemple avec divers caractères :

```
NOBLANK = '[^ ]*'
```

De plus cette valeur, ne doit pas être vide :

```
NOBLANK = '[^ ]+'
```

IPADDR : Une adresse IP se compose de 4 octet ils sont séparés par un « . » point. Un octet peut être un nombre compris entre 0 et 255. Si nous voulons définir le première octet, il faut :

Avoir un chiffre entre 0 et 9 :	[0-9]
un nombre compris entre 10 et 99 :	[1-9][0-9]
un nombre compris entre 100 et 199 :	1[0-9][0-9]
un nombre compris entre 200 et 249 :	2[0-4][0-9]
Avoir un nombre entre 250 et 255 :	25[0-5]

Suite de la solution, nous allons séparer tout simplement par le signe '|' chaque partie de l'adresse-IPv4 avec l'expression : « [0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5] » ainsi nous avons tous les octets. Cela nous permet désormais d'avoir notre adresse-IPv4 de 4 octets séparé par un point (pour écrire un point vous devez placer un *Backslashes* (ou barre oblique inversée) sinon, il remplacera tout caractère). Vous pouvez voir ci-dessous la syntaxe tiré du fichier exp :

```
OCTET = '[0-9]|[1-9][0-9]|1[0-9][0-9]|2[0-4][0-9]|25[0-5] '
IPADDR = '((RE:OCTET)\.){3}(RE:OCTET)'
```

Aide à la conception des expressions régulières

Si vous voulez créer et tester les expressions régulières, vous pouvez utiliser l'outil `regexp` pour l'expression rationnelle, qui est situé dans le répertoire `unix` ou `windows` du paquetage « base ». Avec la syntaxe suivante :

```
utilisation de~: regexp [-c <check dir>] <regexp> <string>
```

Courte explication des paramètres :

- **<check dir>** est le répertoire contenant les fichiers de contrôle et de vérification avec les fichiers exp, ainsi « regexp » pourra recourir au expression déjà définie.
- **<regexp>** est une expression régulière (dans le doute, toujours utiliser les guillemets '...' ou "..." ils sont nécessaires si les apostrophes veulent apparaître dans l'expression)
- **<string>** est la chaîne à examiner

Voici quelques exemples :

```
./i586-linux-regex -c ../check '[0-9]' 0
adding user defined regular expression='[0-9]' ('^([0-9])$')
checking '0' against regexp '[0-9]' ('^([0-9])$')
'[0-9]' matches '0'

./i586-linux-regex -c ../check '[0-9]' a
adding user defined regular expression='[0-9]' ('^([0-9])$')
checking 'a' against regexp '[0-9]' ('^([0-9])$')
regex error 1 (No match) for value 'a' and regexp '[0-9]' ('^([0-9])$')

./i586-linux-regex -c ../check IPADDR 192.168.0.1
using predefined regular expression from base.exp
adding IPADDR='((RE:OCTET)\.){3}(RE:OCTET)'
('^(((1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.){3}(1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]))$')
'IPADDR' matches '192.168.0.1'

./i586-linux-regex -c ../check IPADDR 192.168.0.256
using predefined regular expression from base.exp
adding IPADDR='((RE:OCTET)\.){3}(RE:OCTET)'
('^(((1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5])\.){3}(1?[0-9]?[0-9]|2[0-4][0-9]|25[0-5]))$')
regex error 1 (No match) for value '192.168.0.256' and regexp
'((RE:OCTET)\.){3}(RE:OCTET)'
(unknown:-1) wrong value of variable cmd_var: '192.168.0.256' (invalid ipv4 address)
```

1.3.7 Contrôle détaillé de la configuration

Il est parfois nécessaire d'effectuer des contrôles plus ou moins complexes. Exemple de choses complexe. Avoir une relation entre les paquetages ou avoir une condition à remplir lorsque des variables prennent certaines valeurs. Par ex. lors du choix d'un adaptateur-PCMCIA-ISDN dans le paquetage « pcmcia ».

Pour effectuer ces contrôles, on peut écrire dans le fichier `check/<PAQUETAGE>.ext` (également appelé ext-script) différent petit test. Ce langage se compose des éléments suivants :

1. Mot-clé :

- Contrôle de flux :
 - `if (expr) then statement else statement fi`
 - `foreach var in set_var do statement done`
 - `foreach var in set_var_1 ... set_var_n do statement done`
 - `foreach var in var_n do statement done`
- Relation :
 - `provides package version x.y.z`
 - `depends on package version x1.y1 x2.y2.x2 x3.y3 ...`
- Action :
 - `warning "warning"`
 - `error "error"`
 - `fatal_error "fatal error"`
 - `set var = value`
 - `crypt (variable)`
 - `stat (filename, res)`
 - `fgrep (filename, regex)`

— `split (string, set_variable, character)`

2. Type de donnée : chaîne de caractère, nombre entier positif, numéros de version
3. Opération logique : `<`, `==`, `>`, `!=`, `!`, `&&`, `||`, `~=`, `copy_pending`, `samenet`, `subnet`

Type de données

C'est-à-dire que les types de données de la variable sont associées à une expression régulière et sont affectés en permanence à un type de données :

- Variable dont le type commence par « NUM » numérique et qui contient un nombre entier positif
- Variable, représentant une N-variable pour tout type de liste, elle est également numériquement
- Toutes les autres variables sont traitées comme des chaînes

Cela signifie, entre autres chose, qu'une variable de type ENUMERIC ne peut *pas* être utilisé comme un indice, lors d'un accès à une liste de variables même si vous avez d'abord vérifié si elle n'était pas vide. Le code suivant ne fonctionnera pas comme prévu :

```
# TEST should be a variable of type ENUMERIC
if (test != "")
then
  # Error: You can't use a non-numeric ID in a numeric
  # context. Check type of operand.
  set i=my_array[test]
  # Error: You can't use a non-numeric ID in a numeric
  # context. Check type of operand.
  set j=test+2
fi
```

Voici une solution à ce problème [split](#) (Page 26) :

```
if (test != "")
then
  # all elements of test_% are numeric
  split(test, test_%, ' ', numeric)
  # OK
  set i=my_array[test_%[1]]
  # OK
  set j=test_%[1]+2
fi
```

Substitution de chaîne et de variable

À divers moments les chaînes sont nécessaires, par ex. lorsque vous devez émettre [Attention](#) (Page 22) un avertissement. Certains cas, sont décrits dans cette documentation, une telle chaîne est recherchée pour une variable précise, si elle est trouvée, elle est *remplacé* par son contenu ou par d'autres attributs. Ce remplacement est appelé *substitution de variable*.

Cela va être illustré par un exemple. Supposons cette configuration :

```
# config/base.txt
HOSTNAME='fli41'
# config/dns_dhcp.txt
```

```
HOST_N='1' # Nombre d'hôtes
HOST_1_NAME='client'
HOST_1_IP4='192.168.1.1'
```

Ensuite, les chaînes de caractères sont réécrits comme ceci, si la substitution de variable est actif dans ce contexte :

```
"Mon routeur s'appel $HOSTNAME"
# --> "Mon routeur s'appel fli4l"
"HOSTNAME fait partie du paquetage %{HOSTNAME}"
# --> "HOSTNAME fait partie du paquetage base"
"@HOST_N est $HOST_N"
# --> "Nombre d'hôte est 1"
```

Comme vous pouvez le voir, il y a essentiellement trois options pour le remplacement :

- `$<Name>` ou bien `${<Name>}` : remplace le nom de la variable par le contenu de la variable. C'est la forme la plus courante de substitution. Le nom doit être enfermé dans `{...}` si la chaîne est directement suivie d'un caractère qui peut être une partie valide d'un nom de variable soit une lettre, un chiffre ou un tire souligné. Dans tous les autres cas, l'utilisation des accolades est possible, mais pas obligatoire.
- `%<Name>` ou bien `%{<Name>}` : remplace le nom de la variable par le nom du paquetage, dans lequel la variable est définie. Cela ne fonctionne *pas* dans le script via la variable affectées `set` (Page 22) ou la variable de contrôle de boucle `foreach` (Page 28), étant donné que ces variables ne sont pas dans un paquetage, leur syntaxe est différent.
- `@<Name>` ou bien `@{<Name>}` : remplace le nom de la variable par le commentaire qui est après la variable dans la configuration. Encore une fois, cela n'a pas de sens pour les variables définies par le script.

Remarque : Les éléments de liste de variables ne peuvent *pas* être intégré dans des chaînes de caractères de cette façon, parce qu'il n'y a aucune possibilité de fournir un index.

En général, seule une *constante* peut être utilisé pour la substitution de variable, les chaînes qui proviennent d'une variable restent inchangées. Un exemple permettra de clarifier cela - voici la configuration suivante :

```
HOSTNAME='fli4l'
TEST='${HOSTNAME}'
```

Ensuite, le code :

```
warning "${TEST}"
```

Produit la sortie suivante :

```
Warning: ${HOSTNAME}
```

Et *pas* la sortie :

```
Warning: fli4l
```

Dans les sections suivantes, on notera explicitement les conditions des chaînes qui font l'objet d'une substitution de variable.

Définition d'un service avec un numéro de version associé : **provides**

Cela permet par exemple un OPT peut déclarer et mettre à disposition un serveur d'impression ou un service Web. Un seul paquetage, fourni un certain service. Cela empêche par exemple d'installer deux serveurs web en parallèle, cela ne fonctionnerais pas pour des raisons évidentes, puisque les deux serveurs utiliseraient le port 80. En outre, la version actuelle du service est prévu pour est mise à jour régulièrement. Le numéro de version se compose de deux ou trois nombres séparés par des points, comme « 4.0 » ou « 2.1.23 ».

Les services proviennent généralement de l'OPT, pas du paquetage. Par exemple, dans le paquetage « tools » il y a une série de programmes, qui ont chacun leur propre instruction **provides** elle sera activée via `OPT_...='yes'`.

La syntaxe est :

```
provides <Name> version <Version>
```

Exemple avec le paquetage « easycron » :

```
provides cron version 3.10.0
```

Le numéro de version doit être incrémenté par le développeur de l'OPT dans le troisième volet, si les fonctionnalités ont été seulement améliorations et que l'interface est toujours compatible avec l'OPT. Le numéro de version doit être augmentée avec le premier ou le deuxième chiffre, si l'interface a été modifiée et est incompatible (par exemple, en raison de variables renommées, des chemins changés, manquant ou le programme de service a été renommés, etc.).

Définition une dépendance à un service avec une Version spécifique : **depends**

Si un autre service est nécessaire pour assurer la fonction de votre propre service (par exemple un serveur web), on peut définir une dépendance par une version spécifique pour le service. La version peut être indiquée par deux (par ex. « 2.1 ») ou trois chiffres (par ex. « 2.1.11 »), la version à deux chiffres accepte toutes les versions à partir de ce nombre et la version à trois chiffres accepte seulement la version spécifié. En outre, vous pouvez spécifier une liste de numéros de version si plusieurs versions du service est compatible avec le paquetage.

La syntaxe est la suivante :

```
depends on <Name> version <Version>+
```

Exemple : le paquetage « serveur » contient :

```
provides server version 1.0.1
```

Avec le paquetage « client ». vous indiquez l'instruction **depends** dans cette exemple²

```
depends on server version 1.0      # OK, '1.0' s'adapte '1.0.1'
depends on server version 1.0.1    # OK, '1.0.1' s'adapte '1.0.1'
depends on server version 1.0.2    # Erreur, '1.0.2' s'adapte pas '1.0.1'
depends on server version 1.1      # Erreur, '1.1' s'adapte pas '1.0.1'
depends on server version 1.0 1.1  # OK, '1.0' s'adapte '1.0.1'
depends on server version 1.0.2 1.1 # Erreur, ni '1.0.2' ni '1.1' ne s'adapte '1.0.1'
```

2. bien sûr, un seul à la fois!

Message pour l'utilisateur : warning, error, fatal_error

Avec l'aide de ces trois fonctions, on peut avertir et signaler l'utilisateur, d'une erreur ou interrompre immédiatement le traitement de contrôle. La syntaxe est la suivante :

```
— warning "text"
— error "text"
— fatal_error "text"
```

Toutes les chaînes utilisées avec ces fonctions sont soumis à une [substitution de variable](#) (Page 19).

Affectation

Si vous avez besoin d'utiliser une variable temporaire pour une raison quelconque, vous pouvez la créer avec « `set var [= value]` ». *La variable ne peut pas être une variable de configuration!*³ Si vous omettez « `= value` » la variable sera alors simplement placée sur « `yes` », ensuite vous pouvez tester facilement la variable avec l'instruction `if`. Vous pouvez spécifier les données suivant après le signe égal, variable normale, variable indexée, nombre, chaîne de caractère, versions.

Il convient de noter de par l'assignation le *type* sera défini dans la variable temporaire. Si un numéro est attribué `mkfli4l` se « souviendra » que la variable contient un numéro et permettra plus tard de faire un calcul avec celui-ci. Si vous essayez de faire un calcul avec une variable de type différent, il échouera. Exemple :

```
set i=1    # OK, i est une variable numérique
set j=i+1  # OK, j est une variable numérique et contient la valeur 2
set i="1"  # OK, i est maintenant une variable de chaîne
set j=i+1  # Erreur "Vous ne pouvez pas utiliser un ID non numérique dans un
           #          contexte numérique. Vérifiez le type de l'opérande."
           # --> Pas de calculs avec des chaînes~!
```

Vous pouvez également créer des listes temporaires (voir ci-dessous). Exemple :

```
set prim_%[1]=2
set prim_%[2]=3
set prim_%[3]=5
warning "${prim_n}"
```

Le nombre de liste d'éléments est géré par `mkfli4l` et par la variable `prim_n`. Le code ci-dessus conduit donc à la sortie suivante :

```
Warning: 3
```

Si le côté droit de la cession est une constante de chaîne, elle est soumise à une [substitution de variable](#) (Page 19). L'exemple suivant montre le code :

```
set s="a"
set v1="$s" # v1="a"
set s="b"
```

3. C'est une restriction souhaitée : le contrôle du skript ne sera *pas* en mesure de modifier la configuration de l'utilisateur.

```

set v2="$s" # v2="b"
if (v1 == v2)
then
  warning "égal"
else
  warning "pas égal"
fi

```

La sortie produite n'est « pas égale », parce que les variables `v1` et `v2` sont remplacées par le contenu de la variable `s` déjà en cours de cession.

Remarque : Un ensemble de variables dans un script est visible lors du traitement des autres scripts - actuellement il n'existe pas de principe de localisation pour ces variables introduites. L'ordre dans lequel les scripts sont traités dans divers paquetages, n'est pas définie, vous ne devez pas compter sur une variable ayant des valeurs définies dans un autre paquetage.

Liste (ou Tableau)

Si vous voulez accéder aux éléments d'une %-variable (ou de la liste), vous devez utiliser le nom original de la variable comme mentionné dans le fichier `check/<PAQUETAGE>.txt` et d'ajouter un index pour chaque signe « % » en utilisant « *[Index]* ».

Exemple : Si vous voulez accéder aux éléments de la variable `PF_USR_CHAIN_%_RULE_%` vous avez besoin de deux index car la variable a deux signes « % ». Tous les éléments peuvent être enregistrés par exemple en utilisant la (boucle `foreach` et [voir ci-dessous](#) (Page 28)) :

```

foreach i in pf_usr_chain_n
do
  # un seul index nécessaire, seul un '%' dans la variable
  set j_n=pf_usr_chain_%_rule_n[i]
  # Attention: a
  # foreach j in pf_usr_chain_%_rule_n[i]
  # n'est pas possible, d'où l'utilisation de j_n~!
  foreach j in j_n
  do
    # deux index nécessaires, deux '%' dans la variable
    set rule=pf_usr_chain_%_rule_%[i][j]
    warning "Rule $i/$j: ${rule}"
  done
done

```

Exemple de configuration

```

PF_USR_CHAIN_N='2'
PF_USR_CHAIN_1_NAME='usr-chain_a'
PF_USR_CHAIN_1_RULE_N='2'
PF_USR_CHAIN_1_RULE_1='ACCEPT'
PF_USR_CHAIN_1_RULE_2='REJECT'
PF_USR_CHAIN_2_NAME='usr-chain_b'
PF_USR_CHAIN_2_RULE_N='1'
PF_USR_CHAIN_2_RULE_1='DROP'

```

la sortie suivante est imprimé :

```
Warning: Rule 1/1: ACCEPT
Warning: Rule 1/2: REJECT
Warning: Rule 2/1: DROP
```

Alternativement, vous pouvez parcourir directement toutes les valeurs du tableau, mais les indices exacts ne sont pas toujours connus (car se n'est pas nécessaire) :

```
foreach rule in pf_usr_chain_%_rule_%
do
    warning "Rule %{rule}='${rule}'"
done
```

Cela produit la sortie suivante avec l'exemple de configuration à partir des informations ci-dessus :

```
Warning: Rule PF_USR_CHAIN_1_RULE_1='ACCEPT'
Warning: Rule PF_USR_CHAIN_1_RULE_2='REJECT'
Warning: Rule PF_USR_CHAIN_2_RULE_1='DROP'
```

Le deuxième exemple montre bien le sens de la syntaxe `%<Name>` : la chaîne `%rule` est substitué par le *nom* de la variable en question (par exemple `PF_USR_CHAIN_1_RULE_1`), tandis que `$rule` est substitué par son *contenu* (exemple `ACCEPT`).

Crypter un mot de passe : `crypt`

Dans le fichier `rc.cfg` certaines variables contiennent des mots de passe qui ne doivent pas apparaître en clair. Ces variables peuvent être cryptées en utilisant `crypt` et seront transférées dans un format qui est également utilisé par le routeur. On utilise pour cela :

```
crypt (<variable>)
```

La fonction `crypt` est *seulement* un endroit où la variable de configuration peut être modifiée.

Contrôle des propriétés d'un fichier : `stat`

`stat` vous permet de rechercher les propriétés d'un fichier. Il indique pour l'instant que la taille du fichier. Si vous souhaitez tester des fichiers de configuration dans le répertoire courant, vous pouvez utiliser la variable interne `config_dir`. La syntaxe est :

```
stat (<nom de fichier>, <clé>)
```

La commande ressemble à ceci (les paramètres utilisés ne sont que des exemples) :

```
foreach i in openvpn_%_secret
do
    stat("${config_dir}/etc/openvpn/${i}.secret", keyfile)
    if (keyfile_res != "OK")
    then
        error "OpenVPN: missing secretfile <config>/etc/openvpn/${i}.secret"
    fi
done
```


L'exemple suivant vérifie si un fichier existe dans le répertoire de configuration actuelle. Si la variable `OPENVPN_1_SECRET='test'` est définie dans le fichier de configuration, lors du premier contrôle d'exécution la boucle vérifiera l'existence du fichier `etc/openvpn/test.secret` dans le répertoire de configuration actuelle.

Après l'exécution deux variables sont définies :

- `<Clé>_res` : Résultat après l'exécution système `stat` (« OK », si l'exécution système est réussi, sinon il y aura message d'erreur de l'exécution système)
- `<Clé>_size` : taille du fichier

Cela pourrait alors ressembler à ceci :

```
stat ("unix/Makefile", test)
if ("${test_res}" == "OK")
then
    warning "test_size = $test_size"
else
    error "Error '${test_res}' while trying to get size of 'unix/Makefile'"
fi
```

Un nom de fichier passé comme une constante de la chaîne est soumis à une [substitution de variable](#) (Page 19).

Rechercher dans les fichiers : `fgrep`

Si vous voulez rechercher un fichier avec « `grep` », ⁴ vous avez aussi la possibilité d'utiliser la commande `fgrep`, la syntaxe est :

```
fgrep (<nomdefichier>, <regex>)
```

Si le fichier `<nom de fichier>` n'existe pas alors `mkfli41` s'arrête et affiche une erreur fatale ! Si vous n'êtes pas sûr que le fichier est toujours présent, vous devez avant utiliser la commande `stat` pour savoir si le `<nom de fichier>` existe. Après avoir exécuté `fgrep` le résultat de recherche sera présent dans un tableau et `FGREP_MATCH_%` sera disponible, avec l'indice *x* comme d'habitude vous allez avoir `FGREP_MATCH_N`. `FGREP_MATCH_1` fait référence à toute les lignes l'expression régulière correspondantes, tandis que `FGREP_MATCH_2` pour `FGREP_MATCH_N` contient la *n*ème *n-1* partie entre parenthèses.

Un premier exemple simple vous montrer comment l'utiliser. Le fichier `opt/etc/shells` contient la ligne :

```
/bin/sh
```

Le code est le suivant

```
fgrep("opt/etc/shells", "~/(.)(.*)/")
foreach v in FGREP_MATCH_%
do
    warning "%v='$v'"
done
```

Produit la sortie suivante :

4. « `grep` » est une commande du système d'exploitation Unix, pour les flux de texte de filtrage.

```
Warning: FGREP_MATCH_1='/bin/'
Warning: FGREP_MATCH_2='b'
Warning: FGREP_MATCH_3='in'
```

Le RegEx correspond (seulement) à « /bin/ », (seul) cette partie de ligne est contenue dans la variable `FGREP_MATCH_1`. La première partie entre les parenthèses de l'expression correspond au premier caractère après le premier signe « / », c'est pourquoi « b » est contenu dans `FGREP_MATCH_2`. La deuxième partie restante contient entre les parenthèses « b » jusqu'au dernière signe « / », donc « in » est dans la variable `FGREP_MATCH_3`.

Le deuxième exemple suivant vous montrer une utilisation pratique de `fgrep` avec le fichier `check/base.ext`. Il sera testé si toutes les références `tmpl:` indiquées, sont vraiment présent dans `PF_FORWARD_x` :

```
foreach n in pf_forward_n
do
  set rule=pf_forward_%[n]
  if (rule =~ "tmpl:([[:space:]]+)")
  then
    foreach m in match_%
    do
      stat("$config_dir/etc/fwrules.tmpl/$m", tmplfile)
      if(tmplfile_res == "OK")
      then
        add_to_opt "etc/fwrules.tmpl/$m"
      else
        stat("opt/etc/fwrules.tmpl/$m", tmplfile)
        if(tmplfile_res == "OK")
        then
          add_to_opt "etc/fwrules.tmpl/$m"
        else
          fgrep("opt/etc/fwrules.tmpl/templates", "^$m[[:space:]]+")
          if (fgrep_match_n == 0)
          then
            error "Can't find tmpl:$m for PF_FORWARD_${n}='$rule'!"
          fi
        fi
      fi
    done
  fi
done
```

La valeur du nom de fichier ainsi que l'expression régulière passée comme une constante de chaîne sont soumis à une [substitution de variable](#) (Page 19).

Désassemblage des paramètres : `split`

Souvent plusieurs paramètres sont appliqués dans une variable, ensuite dans le script de démarrage ces paramètres sont désassemblés séparément. Si vous voulez effectuer des tests lors du désassemblage, c'est la commande `split` qu'il vous faut. La syntaxe est :

```
split (<Chaîne>, <Liste>, <Séparateur>)
```

Une chaîne peut être spécifiée par une variable ou directement en tant que constante. `mkfli41` décompose la chaîne là où apparaît un séparateur et génère un élément pour chaque partie de la liste. Vous pouvez parcourir ces éléments plus tard et effectuer des tests. Si rien n'est trouvé entre deux séparateurs un élément de la liste est générée comme une valeur pour une chaîne vide. L'exception « » est : tous les espaces sont supprimés et aucune variable vide n'est créée.

Lors de la décomposition des éléments, si un contexte numérique apparaît dans la variable (par exemple sous forme d'indice) cela doit être précisé dans la commande `split`. Vous devez ajouter un attribut supplémentaire 'numéric'. Cette exécution se présente comme ceci :

```
split (<Chaîne>, <Liste>, <Séparateur>, numeric)
```

Voici un exemple :

```
set bar="1.2.3.4"
split (bar, tmp_%, '.', numeric)
foreach i in tmp_%
do
    warning "%i = $i"
done
```

Produit la sortie suivante :

```
Warning: TMP_1 = 1
Warning: TMP_2 = 2
Warning: TMP_3 = 3
Warning: TMP_4 = 4
```

Remarque : si vous utilisez une variable « numeric » `mkfli41` ne vérifiera *pas* les parties de chaîne générées si elle ne sont pas vraiment numérique ! Si vous utilisez une telle construction dans un contexte numérique `mkfli41` déclenchera une erreur fatale si une telle variable n'est pas numérique. Exemple :

```
set bar="a.b.c.d"
split (bar, tmp_%, '.', numeric)
# Fehler: invalid number 'a'
set i=tmp_%[1]+1
```

Une valeur utilisée dans le première paramètre de la constant de chaîne est soumis à une [substitution de variable](#) (Page 19).

Ajouter des fichiers dans l'archive : `add_to_opt`

Avec la fonction `add_to_opt` des fichiers supplémentaires peuvent être ajoutés dans une archive OPT ou dans le RootFS. Il est possible de sélectionner *tous* les fichiers du sous-répertoire `opt/` ou à partir du répertoire de configuration. Il n'y a pas de restriction sur l'ajout de fichiers dans un paquetage. Si un fichier doit être à la fois dans `opt/` et dans le répertoire de configuration, `add_to_opt` choisira de copier les fichiers dans le répertoire de configuration. La fonction `add_to_opt` est complexe et est en règle général logique, la fonction décide quel fichier sera copié en premier dans l'archive.

La syntaxe est la suivante :

```
add_to_opt <nom de fichier> [<Flags>]
```

Le `Flags` est optionnel. Les valeurs par défaut de la table 1.2 sont utilisés si aucun `Flags` n'est indiqué.

Ci-après un exemple à partir du paquetage « `sshd` » :

```
if (opt_sshd)
then
  foreach pkf in sshd_public_keyfile_%
  do
    stat("$config_dir/etc/ssh/$pkf", publickeyfile)
    if(publickeyfile_res == "OK")
    then
      add_to_opt "etc/ssh/$pkf" "mode=400 flags=utxt"
    else
      error "sshd: missing public keyfile %pkf=$pkf"
    fi
  done
fi
```

Utiliser d'abord `stat` (Page 24) pour vérifier si le fichier existe bien dans le répertoire config. Si le fichier existe, il sera ajouté à l'archive, sinon `mkfli4l` renvoie un message d'erreur.

Remarque : `mkfli4l` vérifie (Page 10) aussi avec la fonction `add_to_opt` si le fichier à copier, se trouve bien dans le répertoire config.

Les noms de fichiers et des `Flags` qui sont utilisés en tant que constantes de chaîne sont soumis à une [substitution de variable](#) (Page 19).

Contrôle de flux

```
if (expr)
then
    statement
else
    statement
fi
```

Un cas classique de restriction, comme nous connaissons. Si la condition est vraie, alors, l'expression `then` est exécutée, si la condition est fausse, l'expression `else` sera exécutée.

Si vous voulez effectuer des tests de `%`-variable, il faudra tester chaque variable. Pour éviter cela, il y a la boucle `foreach` en deux variantes.

1. Itération sur une liste de variables :

```
foreach <contrôle variable> in <Liste-Variable>
do
    <instruction>
done

foreach <contrôle variable> in <liste-Variable-1> <liste-Variable-2> ...
do
    <instruction>
done
```

Cette boucle parcourt tous le tableau de variables spécifié, en commençant par le premier et le dernier élément, le nombre d'éléments de cette liste est extraites du N-variable associée à ce tableau. La contrôle variable prend les valeurs des variables du tableau respectifs. Il est à noter vous pouvez ajouter un processus optionnel pour les variables du tableau si une valeur n'est pas présents dans la configuration, un élément vide sera généré. Vous pourriez en tenir compte dans le script, par exemple comme ceci :

```
foreach i in template_var_opt_%
do
    if (i != "")
    then
        warning "%i is present (%i='$i')"
    else
        warning "%i is undefined (empty)"
    fi
done
```

Comme vous pouvez le voir dans l'exemple, le *nom* des variables du tableau respectives peut être déterminée avec l'instruction `%<contrôle variable>`.

L'instruction dans la boucle ci-dessus peut être l'un des éléments de contrôle ou de fonctions (`if`, `foreach`, `provides`, `depends`, ...).

Si vous souhaitez accéder exactement à un élément du tableau, vous pouvez y remédier en utilisant la syntaxe `<Liste>[<Index>]`. L'index peut être une variable normale, une constante numérique ou encore un tableau indexé.

2. Itération sur N-variables :

```
foreach <contrôle variable> in <N-Variable>
do
    <instruction>
done
```

Cette boucle s'exécute de 1 à la valeur qui est indiqué dans la N-variable. Vous pouvez utiliser contrôle variable dans un tableau de variables indexé. Donc, si vous voulez parcourir non seulement un tableau de variable, mais plusieurs tableaux de variables en même temps, tous contrôlés par la *même* N-variable, vous prenez la variante de boucle et vous utilisez le contrôle variable pour l'indexation de plusieurs tableaux de variables. Exemple :

```
foreach i in host_n
do
    set name=host_%_name[i]
    set ip4=host_%_ip4[i]
    warning "$i: name=$name ip4=$ip4"
done
```

Le résultat du contenu du tableau de la liste `HOST_%_NAME` et de la liste `HOST_%_IP4` pour cet exemple :

```
Warning: 1: name=berry ip4=192.168.11.226
Warning: 2: name=fence ip4=192.168.11.254
Warning: 3: name=sandbox ip4=192.168.12.254
```

Expressions

Une expression est liée à une valeur et un opérateur à une autre valeur. Cette valeur peut être une variable normale, un élément d'un tableau, ou une constante (nombre, chaîne de caractère ou numéro de version). Toutes les constantes de chaîne dans les expressions sont soumises à une [substitution de variable](#) (Page 19).

Les opérateurs vous permettent de fait à peu près tout ce que vous attendez d'un langage de programmation. Un test pour l'égalité de deux variables pourrait donc ressembler à ceci :

```
var1 == var2
"$var1" == "$var"
```

À noter, que la comparaison selon le type de variables se li dans le fichier `check/<PAQUETAGE>.txt` où ils ont été créés. Si l'une des deux variables est [numérique](#) (Page 19), la comparaison se fait sur une base numérique, c'est-à-dire que les chaînes de caractère sont converties en nombres, puis comparées. La comparaison est fondée par une chaîne, si on compare `"05" == "5"` cela donne un résultat « faux », une comparaison `"18" < "9"` donne « vrai » selon l'ordre lexicographique des chaînes de caractère : le chiffre « 1 » précède le chiffre « 9 » dans le jeu de caractères ASCII.

Pour introduire la comparaison des numéros de version de construction vous devez utiliser `numeric(version)`, cela génère la valeur numérique du numéro de version à des fins de comparaison. Ici on applique :

```
numeric(version) := major * 10000 + minor * 1000 + sub
```

« major » représente la première partie, « minor » la deuxième partie et « sub » la troisième partie du numéro de version. Si « sub » est manquant le nombre dans l'addition ci-dessus sera omis (en d'autres termes « sub » sera égale à zéro).

Vous trouverez dans le tableau 1.3 une liste complète de toutes les expressions. « val » indique une valeur de n'importe quel type, « number » une valeur numérique et « string » une chaîne de caractère.

Opérateur « match »

Avec l'opérateur-match `==~` vous pouvez vérifier, si une expression régulière correspond à la valeur d'une variable. En outre, l'opérateur peut également l'utiliser pour extraire une partie de l'expression de la variable. Après avoir appliqué avec succès une expression régulière à une variable, une table `MATCH_%` contiendra toutes les parties trouvées de la variable. Cela pourrait par exemple ressembler à ceci :

```
set foo="foobar12"
if ( foo ==~ "(foo)(bar)([0-9]*)" )
then
    foreach i in match_%
    do
        warning "match %i: $i"
    done
fi
```

Après une exécution de `mkfli41`, cela conduira vers la sortie suivante :

TABLE 1.3 – Expressions logiques

expression	vraie si
id	id == « yes »
val == val	valeurs de type identique sont égaux
val != val	valeurs de type identique sont inégaux
val == number	numérique la valeur de val == nombre
val != number	numérique la valeur de val != nombre
val < number	numérique la valeur de val < number
val > number	numérique la valeur de val > number
val == version	numeric(val) == numeric(version)
val < version	numeric(val) < numeric(version)
val > version	numeric(val) > numeric(version)
val =~ string	val correspond à une chaîne expression régulière
(expr)	l'expression entre les parenthèses est vraie
expr && expr	les deux expressions sont vraies
expr expr	au moins une des deux expressions est vraie
copy_pending(id)	voir la description ci-dessous
samenet (string1, string2)	chaîne1 décrit le même réseau que la chaîne2
subnet (string1, string2)	chaîne1 décrit un sous-réseau de chaîne2

```
Warning: match MATCH_1: foo
Warning: match MATCH_2: bar
Warning: match MATCH_3: 12
```

Lors de l'utilisation de =~ on peut faire référence à toutes les expressions régulières existantes. Si l'on veut par exemple vérifier, si le pilote de la carte Ethernet-PCMCIA a été sélectionné, sans que la variable OPT_PCMCIA soit sur le paramétrée sur « yes », vous pouvez écrire :

```
if (!opt_pcmcia)
then
  foreach i in net_drv_%
  do
    if (i =~ "^(RE:PCMCIA_NET_DRV)$")
    then
      error "If you want to use ..."
    fi
  done
fi
```

Comme démontré dans l'exemple, il est important *d'ancrer* l'expression régulière avec ^ et \$, si vous avez l'intention d'appliquer une expression dans la variable *entière*. Sinon, l'expression renvoie une valeur « vrai » si une *partie* de la variable est couverte par l'expression régulière, ce qui n'est certainement pas souhaitable dans ce cas.

Vérifier le fichier copié, en fonction de la valeur d'une variable : copy_pending

Avec le processus de vérification `copy_pending` vous pouvez vérifier si le fichier a été copié ou non en fonction de la valeur d'une variable. On peut l'utiliser, par exemple pour tester un

pilote spécifié par l'utilisateur, pour savoir s'il existe bien et s'il a bien été copié. `copy_pending` accepte les noms à tester, sous forme d'une variable ou d'une chaîne.⁵ `copy_pending` vérifie si

- la variable est active (si l'opt est chargé, la variable-OPT doit être placé sur « yes »),
- la variable a été référencée dans le fichier `opt/<PAQUETAGE>.txt`, et si
- le fichier a été copié en fonction de la valeur indiquée.

la fonction `copy_pending` renverra « vrai », s'il ne détecte emphpas le fichier a copier lors de la dernière étape, le processus de copie sera donc (encore en « attente »).

Vous pouvez trouver un petit exemple de l'utilisation de toutes ces fonctions dans le fichier `check/base.ext` :

```
foreach i in net_drv_%
do
    if (copy_pending("%i"))
    then
        error "No network driver found for %i='$i', check config/base.txt"
    fi
done
```

Tous les éléments de la liste `NET_DRV_%` seront détectés pour lesquelles aucune copie n'a été faite car il n'existe pas de configuration correspondante dans le fichier `opt/base.txt`.

Comparer des adresses réseaux avec : `samenet` et `subnet`

Pour vérifier la communication entre les réseaux, nous avons besoin d'un test, pour savoir si les deux réseaux sont identiques ou si l'un des deux est un sous-réseau donc différent. Pour cela, vous avez deux fonctions `samenet` et `subnet` qui vous permet de vérifier le réseau.

```
samenet (netz1, netz2)
```

Le retour est « vrai », si les deux réseaux sont identiques, et

```
subnet (netz1, netz2)
```

Le retour est « vrai », si « netz1 » est un sous-réseau de « netz2 ».

Extension du Kernel par ligne de commande

Obligatoire pour certain OPT, elle est utilisée pour rajouter d'autres paramètres dans le Kernel lors du Boot, on peut contrôler la variable `KERNEL_BOOT_OPTION`, pour savoir si les valeurs ont bien été incluses et au cas échéant un message d'avertissement ou d'erreur sera envoyé. Maintenant avec la variable interne `KERNEL_BOOT_OPTION_EXT` vous pouvez ajouter une option nécessaire mais manquante directement dans le script-ext. Un exemple tiré du fichier `check/base.ext` :

```
if (powermanagement =~ "apm.*|none")
then
    if ( ! kernel_boot_option =~ "acpi=off")
```

5. Comme décrit dans la chaîne des objet de substitution de variable, c'est à dire via la [boucle foreach](#) (Page 28) et le [%<Name> de substitution](#) (Page 19) tous les éléments de la liste (ou un tableau) peuvent être examinées.


```

then
    set kernel_boot_option_ext="${kernel_boot_option_ext} acpi=off"
fi
fi

```

Le paramètre « `acpi=off` » sera transmis au Kernel si aucune gestion d'alimentation ou aucun type « d'APM » n'est nécessaire.

1.3.8 Supporte différent choix de version du Kernel

Les différents choix de version du Kernel se distinguent souvent de quelques détails :

- La modification des pilotes disponible, certains ont disparu, d'autres ont été ajoutés.
- Une partie des modules ont un nom différent
- Certains modules ont un aspect différent
- Les modules se trouvent sur d'autres emplacements

Ces différences sont en grande partie traitées automatiquement par `mkfli41`. Pour définir ces modules disponibles, vous pouvez d'une part, les tester et examiner en fonction de la version les ([expressions régulières conditionnelles](#) (Page 15)) d'autre part, cela vous permet avec `mkfli41` d'utiliser le fichier `opt/<PAQUETAGE>.txt` selon la version utilisée. Ils seront ensuite renommés avec un tiret bas `opt/<PAQUETAGE>_<Kernel-Version>.txt`, ainsi, les composants selon la version du Kernel seront séparés les uns des autres. Le fichier du paquetage « base » sera dans le répertoire `opt` :

- `base.txt`
- `base_3_18.txt`
- `base_3_19.txt`

Le premier fichier (`base.txt`) est *toujours* traité. Les deux autres fichiers sont traités si la version du Kernel « 3.18(*) » ou « 3.17(*) » est utilisée. Comme on peut le voir, certaines parties de la version peuvent être omises dans le nom du fichier, si vous avez un groupe de Kernels les numéros de version seront « écrasés ». En supposant que l'on utilise la `VERSION_KERNEL='3.18.9'` les fichiers suivants (si existant) seront lus et traités pour plusieurs installations :

- `<PAQUETAGE>.txt`
- `<PAQUETAGE>_3.txt`
- `<PAQUETAGE>_3_18.txt`
- `<PAQUETAGE>_3_18_9.txt`

1.3.9 Documentation

Les fichiers de la documentation sont placés dans

- `doc/<LANGUE>/opt/<PAQUETAGE>.txt`
- `doc/<LANGUE>/opt/<PAQUETAGE>.html`

Les fichiers HTML peuvent également être divisés, c'est-à-dire être inclus dans chaque OPT différent. Il faut tout de même qu'un `<PAQUETAGE>.html`, soit le fichier référent pour tous les autres. Les modifications d'un paquetage doivent être documentées dans l'un des fichiers suivants :

- `changes/<PAQUETAGE>.txt`

L'ensemble de la documentation, ne doit pas contenir des tabulateurs et doit contenir un maximum de 79 caractères après un retour à la ligne. Vous devez vous assurer que la documentation pourra être lue correctement avec un éditeur sans retour de ligne automatique.

La documentation peut être produite dans le format \LaTeX et ensuite être transformée en format HTML et PDF. À titre d'exemple, cette documentation peut servir à fli4l. Dans la documentation qui se trouve dans le paquetage « template » traite du minimum requis pour les \LaTeX . Vous pouvez voir une brève description générale dans les paragraphes suivantes.

La documentation de fli4l est actuellement disponible dans les langues suivantes : allemande, anglaise ($\langle \text{LANGUE} \rangle = \text{« english »}$) et française ($\langle \text{LANGUE} \rangle = \text{« french »}$). C'est le responsable du paquetage qui prend la décision pour documenter son paquetage dans n'importe quelle langue. Pour plus de clarté, il est recommandé de créer une documentation en allemand et/ou en anglais (idéalement dans les deux langues).

Conditions pour créer une Documentation- \LaTeX

Pour créer des documents à partir des sources- \LaTeX , vous avez certaines exigences à respecter relatives à l'environnement :

- Sous l'environnement Linux/OS X : Pour faciliter la production de la documentation, vous avez le programme Makefile, avec celui-ci toutes les commandes sont automatisées (Cygwin peut également fonctionner, mais n'est pas testée par l'équipe fli4l)
- Installer \LaTeX 2HTML pour les versions HTML
- Bien sûr pour le \LaTeX (« TeX Live » pour Linux/OS X et « MiKTeX » pour Microsoft Windows sont recommandés) avec le programme « pdftex » et les paquets \TeX suivant :
 - L'actuel Script-KOMA (au moins la version 2)
 - Tous les paquets nécessaires pour pdftex
 - Le paquetage décompressé de la documentation fli4l fournit le makefile et les styles- \TeX

Nom de fichier

Les fichiers de la documentation sont désignés selon le schéma suivant :

$\langle \text{PAQUETAGE} \rangle_main.tex$: ce fichier contient la partie principale de la documentation. $\langle \text{PAQUETAGE} \rangle$ indique le nom du paquetage, il doit être écrit (en minuscules).

$\langle \text{PAQUETAGE} \rangle_appendix.tex$: si vous souhaitez ajouter des commentaires supplémentaires au sujet du paquetage, ils seront placés ici dans l'annexe.

Ces fichiers sont stockés dans le répertoire `fli4l/⟨PAQUETAGE⟩/doc/⟨LANGUE⟩/tex/⟨PAQUETAGE⟩`. Vous pouvez voir ci-dessous un exemple du paquetage « sshd » :

```
$ ls fli4l/doc/deutsch/tex/sshd/
Makefile sshd_appendix.tex  sshd_main.tex  sshd.tex
```

Makefile est chargé de générer la documentation, le fichier `sshd.tex` fournit la structure pour la documentation actuelle et son annexe, qui est situé dans les deux autres fichiers. Vous pouvez consulter des exemples dans la documentation du paquetage « template ».

Principes de base du \LaTeX

\LaTeX fonctionne un peu comme HTML « orienté balise », seulement les balises sont appelées ici « commandes », le format est le suivant : `\commande` ou `\begin{environnement} ... \end{environnement}`.

Dans la mesure du possible, vous devez utiliser ces commandes qui accentu *l'importance* du texte et moins sa *présentation*. Il est donc avantageux de les utiliser par exemple.

`\warning{ne_fait_..._pas}`

Au lieu d'utiliser

`\emph{ne_fait_..._pas}`

cette commande.

Chaque commande ou environnement peut absorber plusieurs paramètres supplémentaires, on peut écrire `\commande{paramètre1}{paramètre2}{paramètreN}`.

Certaines commandes ont des paramètres optionnels (au lieu des accolades) pour fermer la commande vous utilisez les crochets : `\kommando[optionalParameter]{parameter1}` ... Habituellement, un seul paramètre optionnel est utilisé, dans des cas plus rares, il peut y en avoir plus.

Dans le document, certains paragraphes sont séparés par des lignes blanches. L^AT_EX gardera ces sauts de ligne pour séparés les paragraphes dans le texte.

Les caractères suivants ont une signification spéciale dans L^AT_EX ils doivent être précédé du caractère `\` dans le texte pour être écrit normalement : `# $ & _ % { }`. Le caractère « `~` » et « `^` », doit être écrit comme ceci : `\verb?~? \verb?^?`.

Les principales commandes L^AT_EX sont expliquées dans la documentation du paquetage « `template` ».

1.3.10 Formats de fichier

Dans le paquetage tous les fichiers texte (la documentation et les scripts d'installation qui sont sur le routeur) doivent être placés au format DOS, c'est à dire avec un CR/LF au lieu d'un simplement fin de ligne LF. Cela garantit que les utilisateurs Windows pourront également lire la documentation avec « Notepad » (ou bloc-notes), et pourront modifier les scripts sous Windows, ensuite ils seront toujours susceptibles de fonctionner sur le routeur. Les Scriptes sont convertis à la construction des archives dans le format utilisé par le routeur (voir la description des flags dans le tableau 1.2).

1.3.11 Développer la documentation

Si vous devez définir un programme pour une nouvelle interface, à partir d'un paquetage et qui sera utilisé par d'autres programmes, la documentation de cette interface, sera séparée du reste de la documentation et se trouvera dans `doc/dev/<PAQUETAGE>.txt`.

1.3.12 Programme-Client

Si vous ajoutez un programme-client pour un paquetage supplémentaire, il sera placé dans le répertoire `windows/` pour les clients-Windows et dans le répertoire `unix/` pour les clients-Unix et Linux.

1.3.13 Code source

Les Programmes personnalisés et les codes sources peuvent être récupéré dans le répertoire `src/<PACKAGE>/`. Le programme peut être construit comme le programme-fli4l, merci de jeter un œil à la documentation du paquetage « `src` » (Page ??).

1.3.14 Les autres fichiers

Tous les fichiers stockés sur le routeur sont dans le répertoire `opt/`. Voici une description :

- Les scripts dans `opt/etc/boot.d` et `opt/etc/rc.d` sont exécutés pour l'amorçage du système
- Les Scripts dans `opt/etc/rc0.d` sont exécutés lors de l'arrêt du système
- Les Scripts dans `opt/etc/ppp` sont exécutés pour appeler et raccrocher une connexion téléphonique par modem
- Les programmes exécutables et les autres fichiers dans `opt/`, sont utilisés en fonction de leurs positions dans le fichier système (par exemple le fichier `opt/bin/busybox` est placé tout en haut dans le répertoire `/bin`)

Les scripts dans `opt/etc/boot.d/`, `opt/etc/rc.d/` et `opt/etc/rc0.d/` sont nommés de façon suivante :

```
rc<numéro>.<nom>
```

Le numéro détermine l'ordre d'exécution de l'installation, le nom donne une indication, pour quel programme/paquetage le script est traité.

1.4 Conditions générales de création de script pour fli4l

Nous n'avons *pas* écrit d'introduction générale pour le Scripts-Shell, mais vous pouvez lire cette introduction sur Internet, ici nous traitons que des situations particulières pour fli4l. Des informations complémentaires sont disponibles dans les différentes pages du manuel Unix-/Linux-. Les liens suivants peuvent servir de point de départ pour ce sujet :

- Introduction au Scripts-Shell :
 - <http://cip.physik.uni-freiburg.de/main/howtos/sh.php>
- Pages d'aide en ligne :
 - <http://linux.die.net/>
 - <http://heapsort.de/man2web>
 - <http://man.he.net/>
 - http://www.linuxcommand.org/superman_pages.php

1.4.1 Structure

Dans le monde Unix, il est essentiel de démarrer le script avec le nom de l'interpréteur de commande, à la première ligne vous devez indiquer :

```
#!/bin/sh
```

Pour que l'on puisse identifier plus facilement le script, à savoir, à quoi il sert, qui l'a écrit, il faut maintenant que celui-ci soit suivi d'une en-tête à peu près comme ceci :

```
#-----  
# /etc/rc.d/rc500.dummy - start my cool dummy server  
#  
# Creation:      19.07.2001  Toller Hecht <toller-hecht@example.net>  
# Last Update:   11.11.2001  Süße Maus <suesse-maus@example.net>  
#-----
```

Vous pouvez maintenant poursuivre le script ...

1.4.2 Gestion des variables de configuration

Généralités

La composition de la configuration de fli4l est dans le fichier `config/<PAQUETAGE>.txt`. Cette documentation contient les [Variables actives](#) (Page 11) et la création du support de boot avec le fichier `rc.cfg`. Lors du boot du routeur, ce fichier est lu avant tous les scripts-rc (les scripts sont dans `/etc/rc.d/`). Ce script peut accéder avec le `$<nom de variable>` à toutes les variables de configuration du routeur.

Avez-vous besoin des valeurs des variables de configuration, même après le boot ? Vous pouvez à partir du fichier `/etc/rc.cfg`, avec lequel vous avez écrit la configuration pour le support de boot. Par exemple, vous pouvez lire la valeur de la variable `OPT_DNS`, avec un script, vous devez le faire de la manière suivante :

```
eval $(grep "^OPT_DNS=" /etc/rc.cfg)
```

Cela fonctionne également avec plusieurs variables (c'est-à-dire en utilisant une seule fois le programme `grep`) :

```
eval $(grep "^\(HOSTNAME\|DOMAIN_NAME\|OPT_DNS\|DNS_LISTEN_N\)=" /etc/rc.cfg)
```

Stockage persistant des données

Les paquetages ont parfois besoin de stocker des données sur un support persistant, ils pourront survivre au redémarrage du routeur. Il existe une fonction `map2persistent`, elle peut être utilisée depuis un script qui sera enregistré dans `/etc/rc.d/`. Ce script doit contenir la variable avec le chemin et le sous-répertoire. L'idée est que la variable est configurée avec un chemin réel – Alors, ce chemin sera utilisé, car l'utilisateur l'a souhaité ou la variable sera configurée sur « auto » – Alors, un sous-répertoire correspondant sera créé en-dessous du répertoire sur un support persistant selon le deuxième paramètre. La fonction retourne le résultat de la variable, avec le nom qui a été indiqué dans le premier paramètre.

Un exemple permettra de clarifier cela. Soit la variable `VBOX_SPOOLPATH`, qui est paramétrée avec un chemin ou avec la valeur « auto ». Pour l'activation

```
begin_script VBOX "Configuring vbox ..."
[...]
map2persistent VBOX_SPOOLPATH /spool
[...]
end_script
```

Cela signifie que la variable `VBOX_SPOOLPATH`, ne sera pas modifiée (si elle contient un chemin) ou le chemin sera remplacé par `/var/lib/persistent/vbox/spool` (Si elle contient la valeur « auto »). La valeur se réfère ⁶ `/var/lib/persistent` est le répertoire pour enregistrer les données sur un support de stockage non volatile, `<SCRIPT>` représente un minuscule script d'exécution (ce nom est dérivé du premier argument [exécute begin_script](#) (Page 38)). Si aucun support convenable n'existe (cela peut être possible), le répertoire `/var/lib/persistent` sera enregistré dans le disque RAM.

Il convient de noter, que le chemin utilisé par `map2persistent` n'est *pas* généré automatiquement – Cela doit être fait soi-même (peut-être avec la commande `mkdir -p <chemin>`).

6. à l'aide d'un soi-disant « lien » monté

Dans le fichier `/var/run/persistent.conf` vous pouvez vérifier si le support de stockage persistant pour les données est possible. Exemple :

```
. /var/run/persistent.conf
case $SAVETYPE in
persistent)
    echo "Stockage persistant possible!"
    ;;
transient)
    echo "Stockage persistant PAS possible!"
    ;;
esac
```

1.4.3 Recherche d'erreur

Au démarrage d'un script, il est souvent utile d'activer le mode débogage, pour déterminer si « un ver est dans le script » et pour savoir s'il est inséré au début ou à la fin du texte :

```
begin_script <OPT-Name> "start message"
<script code>
end_script
```

En fonctionnement normal, un texte apparaît au démarrage du script et à la fin de se même texte le préfixe « finished » sera spécifié.

Si vous voulez déboguer un script, vous devez faire deux choses :

1. Il faut mettre `DEBUG_STARTUP` (Page ??) sur « yes ».
2. Vous devez activer le débogage de l'OPT choisi. On le fait en général par la variable suivante dans les fichiers de configurations : ⁷

```
<OPT-Name>_DO_DEBUG='yes'
```

Maintenant vous pourrez voir sur la console, la représentation exact de l'exécution du programme.

D'autres variables pour le débogage

DEBUG_ENABLE_CORE Si cette variable est placée sur « yes », elle permet de créer un core-Dumps (ou image mémoire). Si un programme se bloque en raison d'une erreur, un fichier image enregistre l'état actuel du système, il pourra ensuite être utilisée pour une analyse du problème. L'image core-Dumps sera enregistrée dans le dossier `/var/log/dumps`.

DEBUG_IP Si cette variable est placée sur « yes », tous les appels du programme par le protocole ip seront enregistrés.

DEBUG_IPUP Si cette variable est placée sur « yes », lors de l'exécution des scripts `ip-up/ip-down` les instructions exécutées seront stockées dans le système de journalisation.

LOG_BOOT_SEQ Si cette variable est placée sur « yes », elle enregistre dans `bootlogd` tous le processus de Boot visible sur la console. Cette variable est placée par défaut sur « yes ».

7. parfois, plusieurs scripts de démarrage sont utilisés pour chaque variable de débogage, ces variables ont des noms différents pour le débogage. Voici un rapide coup d'oeil sur ces scripts.

DEBUG_KEEP_BOOTLOGD Normalement `bootlogd` se termine à la fin du processus de boot. Si on active cette variable, cela permet l'enregistrement au-delà de l'arrêt du processus de boot visible sur la console.

DEBUG_MDEV Si on active cette variable cela génère le protocole Démons-`mdev` et produit un fichier sur tous les périphériques dans le dossier `/dev`

1.4.4 Remarques

- Il est *toujours* préférable d'utiliser les accolades « `{...}` » à la place des parenthèses « `(...)` ». Il convient après l'ouverture de l'accolade de placer, un espace ou un saut de ligne avant la prochaine commande et de placer avant la fermeture de l'accolade un point-virgule ou un nouveau saut de ligne. Par exemple :

```
{ echo "cpu"; echo "quit"; } | ...
```

Équivaut à :

```
{  
    echo "cpu"  
    echo "quit"  
} | ...
```
- Un script peut être arrêté prématurément avec la commande « `exit` ». Mais pour le Script de démarrage (`opt/etc/boot.d/...`, `opt/etc/rc.d/...`), le Script d'arrêt (`opt/etc/rc0.d/...`) et les Scripts `ip-up/ip-down` avec (`opt/etc/ppp/*`) cela est carrément mortelle, il faut dire aussi que ces Scripts ne seront plus exécutés. En cas de doute, ne toucher à rien.
- KISS - Keep it small and simple. Vous voulez utiliser Perl en tant que langage script ? Les possibilités d'écriture pour `fi4l` ne te suffisent pas ? Penser à votre installation ! Votre OPT en a vraiment besoin ? `fi4l` est toujours « uniquement » un routeur, un routeur ne doit pas offrir de services, comme un serveur.
- Le message d'erreur : « `not found` » signifie le plus souvent que le script est encore au format-DOS. Autre problème : si le script n'est pas exécuté. Dans les deux cas, vous devez vérifier le fichier `opt/<PACKAGE>.txt`, pour voir si les options sont correctes (par rapport au « `mode` », « `gid` », « `uid` » et `Flags`). Si le script est produit seulement au démarrage du routeur, vous devez exécuter la commande « `chmod +x <nom du script>` ».
- Pour les fichiers temporaires, vous devez utiliser le chemin `/tmp`. Mais il est essentiel de veiller à ne pas utiliser trop d'espace, parce que le dossier est dans un `Ramdisk-RootFS` ! Si vous avez besoin de plus d'espace, il faut créer un `Ramdisk` et le monter. L'ensemble des détails à ce sujet se trouvent dans le paragraphe "RAM-Disks" de la documentation Dev.
- Afin que les fichiers temporaires obtiennent un nom unique, vous devez ajouter un ID de processus actuelle, dans la variable du Shell le caractère « `$` » sera ajouté au nom du fichier. `/tmp/<OPT-Name>.$$` et le nom du fichier sera correct, mais `/tmp/<OPT-Name>` est plutôt moins, bien sûr `<OPT-Name>` ne doit pas laissé comme ceci, mais vous devez le modifier en fonction.

1.5 Utiliser le filtrage de paquets

1.5.1 Ajouter vos propres chaînes et règles

Un ensemble de routines est fourni pour manipuler le filtrage de paquets, elles permettent d'ajouter ou de supprimer des chaînes (en anglais "Chains") et des règles. Une chaîne sert à

nommée une liste de règles ordonnées. Il y a déjà un ensemble de chaînes prédéfinies sur le routeur fli4l (PREROUTING, INPUT, FORWARD, OUTPUT, POSTROUTING). Vous pouvez créer d'autres chaînes pour certaines fonctions et selon vos besoins.

add_chain/add_nat_chain <chain> : ajoute une chaîne de « filtrage » ou de table « nat »

flush_chain/flush_nat_chain <chain> : supprime toutes les règles d'une chaîne de « filtrage » ou de table « nat »

del_chain/del_nat_chain <chain> : supprime une chaîne à partir du « filtrage » ou de la table « nat ». Les chaînes doivent être vides avant qu'elles puissent être supprimées, et ces chaînes ne doivent pas avoir de référence. Une telle référence est par exemple l'action JUMP dont la cible est précisément cette chaîne.

add_rule/ins_rule/del_rule : ajoute une règle à la fin d'une chaîne (**add_rule**) ou partout dans la chaîne **ins_rule**) ou supprime une règle d'une chaîne (**del_rule**). l'exécution sera la suivante :

```
add_rule <table> <chain> <rule> <comment>
ins_rule <table> <chain> <rule> <position> <comment>
del_rule <table> <chain> <rule> <comment>
```

les paramètres ont les significations suivantes :

table La table dans laquelle la chaîne est insérée

chain La chaîne dans laquelle la règle est insérée

rule La règle qui doit être insérée, vous devez utiliser le même format que dans le fichier-config

position La position dans laquelle la règle doit être insérée (seulement avec **ins_rule**)

comment Le commentaire, il sera affiché avec la règle, il sera vu dans le filtrage de paquets.

1.5.2 Classer les règles dans une infrastructure

fli4l configure les règles du filtrage de paquets avec certaine norme. Si vous souhaitez insérer vos propres règles, vous pouvez ajouter ces règles, après les règles par défaut. Pour cela, vous devez savoir quelle action l'utilisateur a effectué pour rejeter les paquets. Ces informations peuvent être obtenues pour les chaînes FORWARD et INPUT en utilisant deux fonctions, **get_defaults** et **get_count**. Après l'exécution de

```
get_defaults <chain>
```

On obtient les résultats suivants :

drop : cette variable contient la chaîne dans laquelle le paquet sera dévié et rejeté.

reject : cette variable contient la chaîne dans laquelle le paquet sera dévié et refusé.

Après l'exécution de

```
get_count <chain>
```


la variable `res` contient le nombre de règle de la chaîne `<chain>`. Cette position est importante car vous ne pouvez *pas* utiliser simplement `add_rule` pour ajouter une règle à la fin de les chaînes de « filtrage » prédéfini `INPUT`, `FORWARD` et `OUTPUT`. Car ces chaînes sont déjà terminées par des règles par défaut, qui traite tous les paquets restants, en fonction de la disponibilité de la variable `PF_<Chaîne>_POLICY`. Donc si vous insérez cette dernière règle *après* les autres il n'y aura aucun effet. La fonction `get_count` permet maintenant, de détecter l'emplace directement *avant* la dernier règle et de transmettre cette position à la fonction `ins_rule` avec le paramètre `<position>` ainsi, la règle souhaitée sera ajouter à la fin de la chaîne appropriée, cependant devant la dernier règle saisis.

Voici un exemple à partir du script `opt/etc/rc.d/rc390.dns_dhcp` du paquetage « `dns_dhcp` » pour une petit explication :

```
case $OPT_DHCPRELAY in
    yes)
        begin_script DHCRELAY "starting dhcprelay ..."

        idx=1
        interfaces=""
        while [ $idx -le $DHCPRELAY_IF_N ]
        do
            eval iface='$DHCPRELAY_IF_'$idx

            get_count INPUT
            ins_rule filter INPUT "prot:udp if:$iface:any 68 67 ACCEPT" \
                $res "dhcprelay access"

            interfaces=$interfaces' -i '$iface
            idx=`expr $idx + 1`
        done
        dhcprelay $interfaces $DHCPRELAY_SERVER

        end_script
    ;;
esac
```

Ici vous pouvez voir dans le milieu de la boucle l'exécution de `get_count` suivie par l'exécution de la fonction `ins_rule`, entre autres la variable `res` est passé comme paramètre `position`.

1.5.3 Extension pour le test de filtrage de paquets

`fi4l` utilise la syntaxe `match:params` dans les règles de filtrage des paquets, cela permet d'avoir des conditions supplémentaires pour les paquets (voir `mac:`, `limit:`, `length:`, `prot:`, ...). Si vous souhaitez ajouter des tests supplémentaires, vous devez faire comme ci-dessous :

1. Vous devez définir un nom approprié. Ce nom doit commencer par une lettre minuscule entre a-z et ensuite être composé de n'importe quelles lettres et de nombres.

Si vous testez de filtrage de paquets pour les règles IPv6, assurez-vous alors que le nom du test n'est pas un composant d'une adresse IPv6 valide !

2. Créer d'un fichier `opt/etc/rc.d/fwrules-<name>.ext`. Le contenu de ce fichier ressemble à ceci :

```
# IPv4 extension is available
foo_p=yes

# the actual IPv4 extension, adding matches to match_opt
do_foo()
{
    param=$1
    get_negation $param
    match_opt="$match_opt -m foo $neg_opt --fooval $param"
}

# IPv6 extension is available
foo6_p=yes

# the actual IPv6 extension, adding matches to match_opt
do6_foo()
{
    param=$1
    get_negation6 $param
    match_opt="$match_opt -m foo $neg_opt --fooval $param"
}
```

Le test de filtrage de paquets ne doit pas nécessairement être implémenté pour les protocoles IPv4 et IPv6 (bien que cela soit préférable et logique pour les deux protocoles de couche 3).

3. Test de l'extension :

```
$ cd opt/etc/rc.d
$ sh test-rules.sh 'foo:bar ACCEPT'
add_rule filter FORWARD 'foo:bar ACCEPT'
iptables -t filter -A FORWARD -m foo --fooval bar -s 0.0.0.0/0 \
    -d 0.0.0.0/0 -m comment --comment foo:bar ACCEPT -j ACCEPT
```

4. L'extension est incluse et tous les fichiers restants (les composants iptables) sont archivés avec le mécanisme connu.
5. Permet l'extension dans la configuration, il faut ajouter FW_GENERIC_MATCH et/ou FW_GENERIC_MATCH6 dans le fichier-exp, par exemple :

```
+FW_GENERIC_MATCH(OPT_FOO) = 'foo:bar' : ''
+FW_GENERIC_MATCH6(OPT_FOO) = 'foo:bar' : ''
```

1.6 Création d'un CGI pour le paquetage *httpd*

1.6.1 Informations générales sur le serveur Web

Le serveur web, qui est utilisé dans fli4l est un `mini_httpd` de ACME Labs. Les fichiers sources peuvent être téléchargés à partir du site http://www.acme.com/software/mini_httpd/. Cependant, quelques modifications ont été apportées pour fli4l. Les ajustements sont dans le paquetage `src` et dans le répertoire `src/fbr/buildroot/package/mini_httpd`.

1.6.2 Nom du script

Le nom du script doit être significatif, de sorte qu'il soit plus facile à distinguer par rapport aux autres scripts, il ne doit pas avoir de collision de noms avec les différents OPTs.

Les sauts de ligne dans les scripts exécutés sous DOS, seront convertis en sauts de ligne UNIX, vous devez aussi modifier le fichier `opt/<PACKAGE>.txt`, voir Table 1.2 (Page 9).

1.6.3 Configuration du menu

Pour faire une nouvelle entrée dans le menu, vous devez paramétrer le fichier `/etc/httpd/menu`. Ce mécanisme permet de faire des modifications du menu pendant le fonctionnement de l'OPT. Pour cela on utilise le script `/etc/httpd/menu`, il vérifie le format du fichier pour que celui-ci soit toujours consistant. Pour ajouter un nouvel élément dans le menu, vous devez utiliser la commande de la manière suivante :

```
httpd-menu.sh add [-p <priority>] <link> <name> [section] [realm]
```

Si vous indiquez un nom dans `<name>` il sera inclus dans la section, que vous avez indiquée dans `[section]`. Si vous omettez la section, le nom sera par défaut inclus dans la section du "paquetage-OPT". Vous indiquez dans `<link>` la cible du nouveau lien. Dans `<priority>` vous spécifiez la priorité de l'élément du menu dans sa section. S'il n'est pas spécifié, 500 sera utilisé pour la priorité par défaut. La priorité doit être un nombre à trois chiffres. Le lien qui se situe le plus haut dans la section a une priorité la plus faible, si vous voulez un lien le plus bas, vous devez par exemple choisir une priorité de 900. De même, la priorité des données sont triées avec la cible du lien. Dans `[realm]` vous indiquez le domaine pour que l'utilisateur connecté puisse avoir les autorisations nécessaires pour voir l'élément du menu qui sera affiché. S'il `[realm]` n'est pas spécifié, l'élément du menu sera toujours affiché, voir aussi la section « [Droits des utilisateurs](#) » (Page 48).

Exemple :

```
httpd-menu.sh add "Nouveau-fichier.cgi" "Cliquez ici" "Outils" "outils"
```

Cet exemple produit dans la section "Outils" un lien avec le titre "Cliquez ici", la destination du lien sera "Nouveau-fichier.cgi", la section sera créée si elle n'est pas définie.

Vous pouvez également supprimer dans le script un enregistrement de lien du menu :

```
httpd-menu.sh rem <link>
```

Avec cette commande le lien `<link>` qui a été enregistré sera supprimé.

Important: *Si plusieurs enregistrements se réfèrent au même fichier dans le menu, toutes ces enregistrements seront supprimés dans le menu.*

Puisque des sections peuvent aussi avoir des priorités, celles-ci peuvent être créées manuellement. Si une section est créée automatiquement lorsque vous ajoutez une entrée, il recevra automatiquement la priorité 500. Voici la syntaxe pour la création d'une section :

```
httpd-menu.sh addsec <priority> <name>
```

Ici aussi `<priority>` doit contenir un nombre à trois chiffres.

Pour apprendre à configurer judicieusement les priorités, il est intéressant de regarder le fichier `/etc/httpd/menu` pendant l'exécution de `fli4l` les priorités sont dans la deuxième colonne.

Pour être complet, voici un bref commentaire sur le format du fichier menu. Si la commande `httpd-menu.sh` ne suffit pas, vous pouvez sauter ce paragraphe. Le fichier `/etc/httpd/menu` a la structure suivante : il est divisé en quatre colonnes. La première colonne il y a une lettre de code, pour différencier les intitulés et les enregistrements. Dans la deuxième colonne il y a les priorités triées. La troisième colonne contient en intitulé un trait d'union, cela veut dire qu'il n'y a aucune signification pour le titre et de la configuration de la cible du lien. Dans le reste de la ligne se trouve le texte qui apparaîtra plus tard dans le menu.

Avec la lettre d'identification « t », une nouvelle section dans le menu sera installée. Pour configurer une entrée normale dans le menu vous utilisez la lettre d'identification « e ». Un exemple :

```
t 300 - Mon grand OPT
e 200 monopt1.cgi lien pour grand
e 500 monopt1.cgi?plus=oui lien plus pour grand
```

Lors de l'édition de ce fichier vous devez vous assurer que le script `httpd-menu.sh` soit toujours trié avant de sauvegarder le fichier. Que les différentes sections soient triées et que les sections enregistrées soient triées par section l'algorithme de tri exacte peut être repris par `httpd-menu.sh` – Cependant, il est possible d'étendre ce script avec de nouvelles fonctionnalités, pour que tous les modifications du menu se produisent à un emplacement central.

1.6.4 Construction d'un script CGI

L'en-tête

Tous les scripts du serveur Web sont de simples scripts shell (les interpréteurs tels que Perl, PHP, etc, sont beaucoup trop volumineux pour `fli4l`). Le script doit obligatoirement commencer par l'en-tête avec (la référence de l'interpréteur, le nom, une indication sur le scénario, l'auteur, la licence).

Script auxiliaire `cgi-helper`

Tout de suite après l'en-tête vous devez placer le script auxiliaire `cgi-helper` avec la commande suivante :

```
. /srv/www/include/cgi-helper
```

L'espace entre le point et le slash (ou la barre oblique) est important !

Ce script auxiliaire permet diverses fonctionnalités pour simplifier la création des CGIs, il est essentiel à `fli4l`. De plus, avec l'intégration de `cgi-helper` il effectue également des tâches standard, telles que l'analyse syntaxique des variables, qui ont été associées à des formulaires ou des transitions URL ou de l'affichage de la langue pour le texte ou des fichiers CSS.

Le tableau 1.4 donne un aperçu des fonctionnalités des scripts `cgi-helper`.

TABLE 1.4 – Les fonctions pour le script `cgi-helper`

Nom	Fonction
<code>check_rights</code>	Vérification des droits des utilisateurs
<code>http_header</code>	Édition d'une en-tête HTTP standard ou d'une en-tête spécifique, par ex., la manière de télécharger des fichiers
<code>show_html_header</code>	Édition complètes d'une en-tête de page (y compris les en-têtes HTTP et les en-têtes du Menu)
<code>show_html_footer</code>	Édition de la fin d'une page HTML
<code>show_tab_header</code>	Édition d'une fenêtre de contenu pour un tableau
<code>show_tab_footer</code>	Édition de la fin de contenu pour un tableau
<code>show_error</code>	Édition d'une fenêtre pour les messages d'erreur (couleur : rouge)
<code>show_warn</code>	Édition d'une fenêtre pour les messages d'alerte (couleur : jaune)
<code>show_info</code>	Édition d'une fenêtre pour les messages d'informations/réussites (couleur : vert)

Contenu d'un script CGI

Afin d'assurer un aspect homogène et surtout pour la compatibilité avec les futures versions-`fl4l`, il est fortement recommandé d'utiliser les fonctions du script auxiliaire `cgi-helper`, même si nous pouvons théoriquement générer toutes les sorties dans un même CGI.

Un simple script-CGI peut ressembler à ceci :

```
#!/bin/sh
# -----
# Header (c) Autor Datum
# -----
# get main helper functions
. /srv/www/include/cgi-helper

show_html_header "Mon premier CGI"
echo '    <h2>Bienvenue</h2>'
echo '    <h3>Ceci est un exemple de script-CGI</h3>'
show_html_footer
```

Fonction `show_html_header`

La valeur qui est indiqué dans la fonction `show_html_header`, est utilisé pour le titre. Il sera généré automatiquement dans le menu, il inclue aussi automatiquement le script CSS et le fichier langue. Il faut pour cela que les fichiers scripts ont le même nom et qu'ils soient dans les répertoires `/srv/www/css` et `/srv/www/lang` (bien sûr, avec une extension différente). Exemple :

```
/srv/www/admin/OpenVPN.cgi
/srv/www/css/OpenVPN.css
/srv/www/lang/OpenVPN.de
```

L'utilisation du fichier-langue ainsi que le fichier-CSS sont facultatif. Intégré toujours `css/nom.css` et `lang/nom.<lang>`, `<lang>` correspond à la langue choisie.

Vous pouvez placer à côté du titre de la fonction `show_html_header` d'autres paramètres. Une exécution de la fonction avec tous les paramètres pourrait ressembler à ceci :

```
show_html_header "Titre" "refresh=$time;url=$url;cssfile=$cssfile;showmenu=no"
```

Tous les paramètres supplémentaires, comme le montre l'exemple ci-dessus, doivent avoir des guillemets et être séparés par un point-virgule. Autrement le contrôle de syntaxe ne sera *pas* réussi ! Il est nécessaire de respecter la syntaxe des paramètres.

Voici un bref aperçu des fonctions de ces paramètres :

- **refresh=time** : Temps en secondes dans lequel la page sera rechargée par le navigateur.
- **url=url** : L'URL est rechargé après un rafraîchissement.
- **cssfile=cssfile** : Nom du fichier CSS, si celui-ci est différent du CGI.
- **showmenu=no** : L'affichage du menu et l'en-tête peut être supprimée.

D'autres conseils pour le contenu du CGI :

- Prenez votre temps :-)
- Écrivez proprement le HTML (SelfHTML ⁸ est un bon point de départ)
- Renoncer au bric-à-brac très moderne, (le JavaScript est OK, si cela ne crée pas de problème avec les utilisateurs, le tout fonctionnent également sans JavaScript)

Fonction `show_html_footer`

La fonction `show_html_footer` ferme le bloc dans le script CGI, qui a été lancé par la fonction `show_html_header`.

Fonction `show_tab_header`

Dans le CGI, pour que le contenu de vos résultats soit bien rangé et affiché sur la page Web, avec l'aide du `cgi-helper` vous pouvez utiliser la fonction `show_tab_header`. Il peut générer des titres dans des onglets cliquables dans un 'Tableau', ainsi votre page peut être divisé en plusieurs zones logiquement séparées.

Les paramètres dans la fonction `show_tab_header` sont toujours écrits deux par deux. Le premier paramètre correspond au titre de l'onglet et le deuxième correspond au lien. Si vous indiquez 'no' comme deuxième paramètre, le titre sera seulement placé dans l'onglet (en couleur) et le lien ne sera pas cliquable.

Dans l'exemple suivant, nous allons créer une 'fenêtre' dans laquelle nous allons mettre le titre 'Une grande fenêtre'. Dans la fenêtre sera écrit 'foo bar' :

```
show_tab_header "Une grande fenêtre" "no"
echo "foo"
echo "bar"
show_tab_footer
```

Dans l'exemple suivant, deux onglets cliquables sont générés, la variable `action` dans le script va transmettre des valeurs différentes.

```
show_tab_header "1. onglet" "$myname?action=machdies" \
                "2. onglet" "$myname?action=machjenes"
echo "foo"
echo "bar"
show_tab_footer
```

8. voir <http://de.selfhtml.org/>

Maintenant le contenu du script peut exploiter la variable `FORM_action` (voir ci-dessous pour l'exploitation de la variable) en fonction des différents paramètres. l'onglet apparaîtra et l'on pourra le sélectionner en cliquant dessus, si vous ne voulez pas que cet onglet soit sélectionnable, vous devez placer dans la fonction 'no', pour ce passé du lien comme indiqué plus haut. Cela sera plus facile, si vous utilisez l'exemple suivant :

```
_opt_machdies="1. onglet"
_opt_machjenes="2. onglet"
show_tab_header "$_opt_machdies" "$myname?action=opt_machdies" \
                "$_opt_machjenes" "$myname?action=opt_machjenes"
case $FORM_action in
    opt_machdies) echo "foo" ;;
    opt_machjenes) echo "bar" ;;
esac
show_tab_footer
```

Si vous utilisez une variable pour sélectionner un titre, ce nom correspond à l'action de la variable et commencera par un tiret (_), ainsi l'onglet correspond à ce nom pourra être sélectionné.

Fonction `show_tab_footer`

La fonction `show_tab_footer` ferme le bloc dans le script CGI, qui a été lancé par la fonction `show_tab_header`.

Supporte le multi-language

Le script `cgi-helper` contient également une fonction pour faire des scripts CGI en utilisant d'autres langues. Pour ce faire, vous devez 'juste' utiliser les variables commençant par un tiret bas (_) pour traduire le texte et de définir ces variables dans la langue de votre choix.

Exemple :

lang/opt.de contient :

```
_opt_machdies="Eine Ausgabe"
```

lang/opt.en contient :

```
_opt_machdies="An Output"
```

admin/opt.cgi contient :

```
...
echo $_opt_machdies
...
```

Utilisation de formulaires

Afin de traiter des formulaires, il faut savoir certaines choses. L'utilisateur doit choisir la méthode de formulaire GET ou POST, les paramètres peuvent être trouvées après le montage du script `cgi-helper` (qui à son tour appelle le de programme auxiliaire `proccgi`) avec la

nouvelle variable `FORM_<Paramètre>`. Si vous avez entré un nom, une "adresse" dans le champ de formulaire et si vous indiquez dans le script CGI `$FORM_adresse`, cette adresse sera accessible.

Pour plus d'informations sur le programme `proccgi` vous pouvez les trouver ici <http://www.fpx.de/fp/Software/ProcCGI.html>.

Droits des utilisateurs : Fonction `check_rights`

Afin de vérifier si l'utilisateur a les droits suffisants pour utiliser un script CGI, la fonction `check_rights` est appelée au début du script CGI comme ceci :

```
check_rights <Section> <Action>
```

Le script CGI ne sera exécuté que si l'utilisateur est enregistré.

- A tous les droits (`HTTPD_RIGHTS_x='all'`), ou
- tous les droits dans un domaine spécifique (`HTTPD_RIGHTS_x='<Domaine>:all'`), ou
- a le droit d'effectuer une action spécifique dans un domaine spécifique (`HTTPD_RIGHTS_x='<Domaine>:<Action>'`).

Fonction `show_error`

Cette fonction renvoie un message d'erreur dans une fenêtre rouge. Il a besoin de deux paramètres : un titre et un message. Exemple :

```
show_error "Error: No key" "No key was specified!"
```

Fonction `show_warn`

Cette fonction renvoie un message d'avertissement dans une fenêtre jaune. Il a besoin de deux paramètres : un titre et un message. Exemple :

```
show_info "Warnung" "Actuellement, il n'y pas de connexion!"
```

Fonction `show_info`

Cette fonction renvoie un message de réussite ou d'information dans une fenêtre verte. Il a besoin de deux paramètres : un titre et un message. Exemple :

```
show_info "Info" "Action a été exécutée avec succès!"
```

Script auxiliaire `cgi-helper-ip4`

Immédiatement après le script `cgi-helper`, vous devez ensuite intégrer le script auxiliaire `cgi-helper-ip4` avec la commande suivante :

```
. /srv/www/include/cgi-helper-ip4
```

L'espace entre le point et le slash (ou la barre oblique) est important !

Ce script fournit des fonctions pour vous aidez à effectuer des tests d'adresse IPv4.

Fonction ip4_isvalidaddr

Cette fonction vérifie si une adresse IPv4 qui a été enregistrée est valide. Exemple :

```
if ip4_isvalidaddr ${FORM_inputip}
then
    ...
fi
```

Fonction ipv4_normalize

Cette fonction supprime les zéros inutile dans l'adresse IPv4 enregistrée. Exemple :

```
ip4_normalize ${FORM_inputip}
IP=$res
if [ -n "$IP" ]
then
    ...
fi
```

Fonction ipv4_isindhcprange

Cette fonction vérifie si l'adresse IPv4 enregistrée est dans la plage d'adresse de début et de fin. Exemple :

```
if ip4_isindhcprange $FORM_inputip $ip_start $ip_end
then
    ...
fi
```

1.6.5 Divers

Des choses et d'autres (et oui, c'est aussi important!) :

- Le `mini_httpd` ne protège pas les sous-répertoires par mot de passe. Vous devez avoir un répertoire `.htaccess` pour chaque utilisateur ou créer un lien vers un autre fichier `.htaccess`.
- KISS - Keep it simple, stupid!
- Ces informations peuvent être changées à tout moment et sans préavis!

1.6.6 Dépannage

Pour faciliter le débogage d'un script CGI, vous pouvez activer le mode débogage avant l'intégration du script `cgi-helper`. Pour cela, la variable `set_debug` doit être paramétrée sur "yes". Cela conduit à la création du fichier `debug.log`, que vous pourrez télécharger sur l'URL `http://<fili41-Host>/admin/debug.log`. Cela inclut tous les appels vers ce script CGI. La variable `set_debug` n'est pas globale, et doit être renouvelée dans tous les scripts CGI en question. Exemple :

```
set_debug="yes"
. /srv/www/include/cgi-helper
```

En outre, cURL⁹ est idéal pour le dépannage, en particulier lorsque les en-têtes HTTP ne sont pas assemblés correctement ou que le navigateur affiche une page blanche. Et aussi, le comportement de mise en cache des navigateurs modernes est un véritable problème.

Exemple : Avec l'exécution suivante, en-tête HTTP ("dump", -D) l'affichage normal du CGI `admin/mon.cgi` sera publié. Vous pouvez utiliser le nom d'utilisateur ("user", -u) "admin".

```
curl -D - http://fli4l/admin/mon.cgi -u admin
```

1.7 Démarrer, arrêter, se connecter et se déconnecter avec fli4l

1.7.1 Concept de Boot

fli4l 2.0 peut être installé sur différents médias, disque-dur ou carte-Compact-Flash(TM), il est aussi possible de l'installer sur un support Zip ou sur un CD-ROM amovible. En outre, l'installation d'une version sur un disque-dur n'est pas fondamentalement différente que sur une disquette.¹⁰

Ces exigences ont été réalisées grâce à l'archive `opt.img`, jusqu'ici il était installé sur un disque-RAM maintenant il peut être placé sur d'autres médias. Il peut s'agir d'une partition sur un disque dur ou une carte-CF. Pour ce second volume, le répertoire `/opt` sera monté et les programmes auront des liens symboliques et seront intégrés dans le rootfs. La structure apparaissant dans le système de fichiers RootFS correspond au répertoire `opt` de la distribution fli4l à une exception près – le préfixe des fichiers seront omis. Le fichier `opt/etc/rc` se trouve directement dans `/etc/rc` et pour le fichier `opt/bin/busybox` il est dans `/bin/busybox`. Ces fichiers sont seulement des liens dans le média monté en lecture seule, on peut les ignorer, tant que les fichiers ne sont pas modifiés. Si vous voulez les modifier, il faut d'abord rendre ces fichiers accessibles en écriture avec la commande `mk_writable` (voir ci-dessous).

1.7.2 Scripts de démarrage et d'arrêt

Les Scripts qui sont exécutés lors du boot système, sont dans les répertoires `opt/etc/boot.d` et `opt/etc/rc.d` ils sont également exécutés dans l'ordre.

Important: *Quand ces scripts sont exécutés aucun processus particulier n'est produit, ils ne peuvent pas être arrêtés avec la commande « exit ». Cette commande conduirait à une rupture du processus de Boot !*

Scripts de démarrage dans `opt/etc/boot.d/`

Les scripts de ce répertoire sont exécutés les premiers. Ils ont pour mission, de monter le périphérique de boot, le fichier de configuration `rc.cfg` se trouve sur le support de boot et décompresse l'archive `opt.img`. Selon le type de Boot (Page ??) il est plus ou moins complexe et font les choses suivantes :

- Charge les pilotes du matériel (optionnel)
- Monte le volume de boot (optionnel)

9. voir <http://de.wikipedia.org/wiki/CURL>

10. À l'origine fli4l pouvait s'exécuter à partir d'une disquette. Depuis fli4l est devenue trop volumineuse, la disquette ne peut plus être utilisée.

- Le fichier de configuration `rc.cfg` est lu depuis le volume de boot et sera écrit dans `/etc/rc.cfg`
- Monte le volume Opt (optionnel)
- Extraît les archives Opt (optionnel)

Lors de la construction des scripts, vous avez la chance d'en apprendre davantage sur la configuration de fli4l, le fichier de configuration est également intégré dans l'archive rootfs, dans ce fichier `/etc/rc.cfg` vous trouverez les variables de configuration qui seront analysées puis les scripts de démarrage seront exécutés depuis le répertoire `opt/etc/boot.d/`. Après le montage du volume de boot, le fichier `/etc/rc.cfg` est remplacé par le fichier de configuration dans le volume de boot, de sorte que les scripts de démarrage dans `opt/etc/rc.d/` soit disponible pour l'actuel configuration du volume de boot (voir ci-dessous).¹¹

Scripts de démarrage dans `opt/etc/rc.d/`

C'est les commandes qui sont exécutées à chaque démarrage du routeur, elles peuvent être stockés dans le répertoire `opt/etc/rc.d/`. Les conventions suivantes s'appliquent :

1. Il faut classer les noms de script comme ceci :

`rc<nombre à trois chiffres>.<nom de l'OPT>`

Les scripts sont démarrés dans l'ordre croissant des numéros. Si plusieurs scripts ont le même numéro attribué, c'est le caractère alphabétique après le point qui détermine l'ordre. L'installation des paquetages s'effectue les uns après les autres, ils sont définis par un numéro.

Voici une estimation approximative, des numéros pouvant être utilisé pour une installation :

Numéro	Fonction
000-099	Système de base (hardware, fuseau horaire, système de fichiers)
100-199	Module Kernel (drivers)
200-299	Connexions externe (PPPoE, ISDN4Linux, PPTP)
300-399	Réseau (routage, interface, filtrage de paquet)
400-499	Serveur (DHCP, HTTPD, Proxy, etc.)
500-900	Tout le reste
900-997	Tout ce qui peut générer un dial-up
998-999	Réservé (ne pas utiliser!)

2. Vous devez *placer* dans ces scripts, toutes les fonctions nécessaires pour changer le RootFS. (par ex. pour la création d'un répertoire `/var/log/lpd`).
3. Vous ne *devez* pas effectuer d'écriture dans les fichiers scripts qui font partie de l'archive-opt, car ces fichiers sont en lecture seule sur le média. Pour modifier de tel fichier, il faut, au préalable rendre accessible ce fichier en écriture via `mk_writable` (voir ci-dessous). En exécutant cette fonction, le fichier si nécessaire, sera copié et sera accessible en écriture sur le RootFS. Si le fichier est déjà en écriture, rien ne se passera lors de l'exécution de la fonction `mk_writable`.

11. Normalement, ces deux fichiers sont identiques. Un changement n'est possible que si le fichier de configuration sur le volume de démarrage a été modifié manuellement, par exemple pour modifier une configuration qui sera utilisé plus tard, sans avoir à reconstruire l'archive fli4l.

Important: *mk_writable* doit être utilisé sur les fichiers qui sont dans le rootfs et non pas par le biais du dossier */opt*. Si vous voulez modifier */usr/local/bin/foo* vous exécutez *mk_writable /usr/local/bin/foo*.

4. Ces scripts ont besoin de vérifiés, avant d'exécuter les commandes réelles, si l'OPT correspondant est actif. Cela se fait habituellement par une simple distinction de cas :

```
if [ "$OPT_<OPT-Name>" = "yes" ]
then
    ...
    # ici OPT start!
    ...
fi
```

5. Pour pouvoir déboguer plus facile, vous devez insérer dans le script les fonctions **begin_script** et **end_script** :

```
if [ "$OPT_<OPT-Name>" = "yes" ]
then
    begin_script F00 "configuring foo ..."
    ...
    end_script
fi
```

Pour juste déboguer de script au démarrage, vous devez simplement activer la variable **FOO_DO_DEBUG='yes'**.

6. Toutes les variables de configuration sont directement disponible par les scripts. Des explications pour accéder à d'autres scripts à partir des variables de configuration peuvent être trouvés dans la section « [Gestion des variables de configuration](#) » (Page 37)
7. Le dossier */opt* ne doit pas être utilisé comme stocker des données OPTs. Si de l'espace supplémentaire est nécessaire, l'utilisateur doit de définir un chemin approprié en utilisant la variable de configuration. Selon le type de données à stocker (données persistante ou transitoire) vous devez utiliser différentes affectations par défaut. Le chemin */var/run/* est logique pour les données transitoires, tandis que pour les données persistant, il est conseillé d'utiliser la fonction [map2persistent](#) (Page 37) combiné avec une variable de configuration appropriée.

Scripts d'arrêt dans *opt/etc/rc0.d/*

Chaque ordinateur a besoin d'un temps pour s'arrêter ou redémarrer. Il se pourrait bien que vous pouvez avoir à effectuer des opérations avant que l'ordinateur s'éteigne ou redémarre. Les commandes officielles sont « halt » et « reboot ». Ces commandes sont également placées dans IMONC ou dans le Web-GUI si vous l'avez installé, un clique sur le bouton suffira pour arrêter ou redémarrer le routeur.

Tous les scripts d'arrêt se trouvent dans le répertoire *opt/etc/rc0.d/*. Le nom du fichier est analogue à celui du script de démarrage. Ils sont également exécutés dans ordre *croissant* des numéros.

1.7.3 Fonctions auxiliaires

Dans */etc/boot.d/base-helper* différentes fonctions sont mises à disposition, elles peuvent être utilisées pour les scripts de démarrage. Cela se applique pour certaines choses comme

pour un support de débogage, pour le chargement des modules du Kernel ou pour la sortie des messages.

Les différentes fonctions sont les suivantes et brièvement décrites.

Contrôle des scripts

begin_script <Symbol> <Message> : envoie un message et active le débogueur de script en utilisant **set -x**, si <Symbol>_DO_DEBUG est sur « yes ».

end_script : envoie un message final et fournit l'état de débogage si vous avez activé le **begin_script**. Pour chaque activation du **begin_script** un **end_script** sera associé et activé (et vice versa).

Chargement des modules du Kernel

do_modprobe [-q] <Module> <Paramètre>* : Charge le module du Kernel, y compris ses paramètres, en résolvant en même temps les dépendances du module. Le paramètre « -q » empêche qu'un message erreur soit mis. La fonction retourne en cas de succès une valeur nulle, dans le cas d'une erreur une valeur non nulle. Cela permet décrire un code pour gérer les échecs lors du chargement les modules du Kernel :

```
if do_modprobe -q acpi-cpufreq
then
    # pas de contrôle de fréquence du CPU via ACPI
    log_error "le contrôle de fréquence du CPU via ACPI n'est pas disponible."
    # [...]
else
    log_info "le contrôle de fréquence du CPU via ACPI est activé."
    # [...]
fi
```

do_modprobe_if_exists [-q] <Chemin> <Module> <Paramètre>* : vérifie d'abord si le module /lib/modules/<version du kernel>/<Chemin>/<Module> existe et ensuite exécute **do_modprobe**.

Important: *Le module doit exister précisément par ce nom, aucun alias ne peuvent être utilisé. Lorsque vous utilisez un alias **do_modprobe** sera exécuté immédiatement.*

Messages et gestion des erreurs

log_info <Message> : envoie un message sur la console et dans /bootmsg.txt. Si aucun message n'est enregistré dans ce paramètre le fichier **log_info** sera lu depuis l'entrée par défaut. La fonction renvoie toujours en retour une valeur nulle.

log_warn <Message> : envoie un message d'avertissement sur la console et dans /bootmsg.txt, la chaîne **WARN:** sera utilisée comme préfixe. Si aucun message n'est enregistré dans ce paramètre le fichier **log_warn** sera lu depuis l'entrée par défaut. La fonction renvoie toujours en retour une valeur null.

log_error <Message> : envoie un message d'erreur sur la console et dans /bootmsg.txt, la chaîne **ERR:** sera utilisée comme préfixe. Si aucun message n'est enregistré dans ce paramètre le fichier **log_error** sera lu depuis l'entrée par défaut. La fonction renvoie toujours en retour une valeur null.

set_error <Message> : envoie un message d'erreur qui sera défini dans une variable d'erreur interne, ensuite elle pourra être examinée via **is_error**.

is_error : réinitialise la variable d'erreur interne et renvoie true, si elle a été précédemment défini par **set_error**.

Fonctions réseau

translate_ip_net <Valeur> <Nom de Variable> [<Variable de Résultat>] :
remplace les références symboliques par des paramètres. Actuellement les traductions suivantes ont lieu :

.*.*., **none**, **default**, **pppoe** ne sont pas remplacés

any sera remplacé par 0.0.0.0/0

dynamic sera remplacée par une adresse IP du routeur, à travers laquelle il existe une connexion Internet.

IP_NET_x sera remplacé par le réseau se trouvant dans la configuration.

IP_NET_x_IPADDR sera remplacé par l'adresse IP se trouvant dans la configuration.

IP_ROUTE_x sera remplacé par le réseau routé se trouvant dans la configuration

@<Hostname> sera remplacé par l'hôte ou par l'adresse IP se trouvant dans la configuration.

Le résultat de la traduction est mémorisée dans la variable dont le nom est transmis dans le troisième paramètre, si ce paramètre est manquant, le résultat sera stocké dans la variable **res**. Le nom de la variable qui est transmis dans le second paramètre est utilisé uniquement pour les messages d'erreur si la traduction échoue, cela permet d'appeler la source de la valeur à traduire. Si une erreur se produit, un message est alors exécuté.

Unable to translate value '<Valeur>' contained in <Nom de Variable>.

La valeur nulle est retournée si la traduction a réussi et une valeur non nulle sera retournée si une erreur s'est produite.

Divers

mk_writable <Fichier> : pour vous assurer que le fichier transmis sera accessible en écriture. Seulement si le fichier est en lecture seule dans le fichier-système et juste monté par un lien symbolique, une copie locale sera alors créée pour avoir le fichier en écrit.

list_unique <Liste> : supprime les doublons dans la liste transmise. Le résultat est écrit dans la sortie standard.

1.7.4 Règles mdev

Il est possible d'établir des règles mdev supplémentaires qui exécutent des actions spécifiques pour l'apparition ou la disparition de certains dispositifs dans les paquetages OPT. Par exemple la variable **OPT_AUTOMOUNT** dans le paquetage **hd**, utilise une telle règle pour monter automatiquement un nouveau support de stockage émergentes. Si vous souhaitez intégrer une règle mdev supplémentaire, vous pouvez utiliser le script sous la forme :

```
/etc/mdev.d/mdev<Nummer>.<Name>
```

Installation du rootFS, le nombre doit être similaire au début et à la fin du script et constitué de trois chiffres, le nom peut être choisi arbitrairement. Dans ce scénario, toutes les sorties sont intégrés dans le fichier standard `/etc/mdev.conf`. Exemple avec `OPT_AUTOMOUNT` mentionné ci-dessus :

```
#!/bin/sh
#-----
# /etc/mdev.d/mdev500.automount - mdev HD automounting rules      __FLI4LVER__
#
#
# Last Update:  $Id: dev_main_boot_dial.tex 51849 2018-03-06 14:55:21Z kristov $
#-----

cat <<"EOF"
#
# mdev500.automount
#

-SUBSYSTEM=block;DEVTYPE=partition;.+      0:0 660 */lib/mdev/automount

EOF
```

L'en-tête de la syntaxe des règles du fichier `/etc/mdev.conf` et la documentation mdev, se trouve sur la page Web <http://git.busybox.net/busybox/plain/docs/mdev.txt> tout y est référencées. Si le script invoque une règle (comme dans l'exemple ci-dessus `/lib/mdev/automount`), il déclenchera un accès à toutes les variables "événements" du kernel et en particulier :

- **ACTION** (Typiquement **add** ou **remove**, plus rarement **change**)
- **DEVPATH** (le chemin sysfs pour le composant concerné)
- **SUBSYSTEM** (le sous-système du kernel affecté, voir ci-dessous)
- **DEVNAME** (le fichier de périphérique affecté dans `/dev`, si manquant, le périphérique ne sera pas créé ou supprimé, mais par exemple un module)
- **MDEV** (est fixé par mdev le nom du fichier du périphérique sera finalement généré)

Exemple de sous-systèmes du Kernel :

block Bloque les périphériques (carte mémoire) comme **sda** pour le (premier disque dur), **sr0** pour le (premier lecteur de CD) ou **ram1** pour le (deuxième disque RAM)

input Périphériques d'entrée de clavier, de la souris, etc., comme le fichier `input/event0` qui correspond à un périphérique, il doit être défini avec sysfs c'est un système de fichiers virtuel.

mem Périphériques pour accéder aux ports de la mémoire et du matériel, comme **mem** et **ports**, il peut également comprendre les pseudo-périphériques comme **zero** (délivre en permanence le caractère ASCII par la valeur null) et **null** (pas de retours, tout sera absorbé)

sound Divers Périphériques pour les sorties audio, appellation hétérogène

tty Périphériques pour l'accès aux consoles physiques ou virtuels, comme **tty1** (première console virtuelle) ou **ttyS0** (première console série)

Un exemple pour les deux premiers ports série :

```
mdev[42]: 30.050644 add@/devices/pnp0/00:04/tty/ttyS0
mdev[42]: ACTION=add
mdev[42]: DEVPATH=/devices/pnp0/00:04/tty/ttyS0
mdev[42]: SUBSYSTEM=tty
mdev[42]: MAJOR=4
mdev[42]: MINOR=64
mdev[42]: DEVNAME=ttyS0
mdev[42]: SEQNUM=613

mdev[42]: 30.051477 add@/devices/platform/serial8250/tty/ttyS1
mdev[42]: ACTION=add
mdev[42]: DEVPATH=/devices/platform/serial8250/tty/ttyS1
mdev[42]: SUBSYSTEM=tty
mdev[42]: MAJOR=4
mdev[42]: MINOR=65
mdev[42]: DEVNAME=ttyS1
mdev[42]: SEQNUM=614
```

Un exemple pour connecter un clavier MF II :

```
mdev[41]: 4.030653 add@/devices/platform/i8042/serio0/input/input0
mdev[41]: ACTION=add
mdev[41]: DEVPATH=/devices/platform/i8042/serio0/input/input0
mdev[41]: SUBSYSTEM=input
mdev[41]: PRODUCT=11/1/1/ab41
mdev[41]: NAME="AT Translated Set 2 keyboard"
mdev[41]: PHYS="isa0060/serio0/input0"
mdev[41]: PROP=0
mdev[41]: EV=120013
mdev[41]: KEY=4 2000000 3803078 f800d001 feffffdf ffffffff ffffffff ffffffff
mdev[41]: MSC=10
mdev[41]: LED=7
mdev[41]: MODALIAS=input:b0011v0001p0001eAB41-e0,1,4,11,14,k71,72,73,74,75,76,77,79,
7A,7B,7C,7D,7E,7F,80,8C,8E,8F,9B,9C,9D,9E,9F,A3,A4,A5,A6,AC,AD,B7,B8,B9,D9,E2,ram4,l0,
1,2,sw
mdev[41]: SEQNUM=604
```

Un exemple pour charger un module kernel USB (uhci_hcd) :

```
mdev[41]: 6.537506 add@/module/uhci_hcd
mdev[41]: ACTION=add
mdev[41]: DEVPATH=/module/uhci_hcd
mdev[41]: SUBSYSTEM=module
mdev[41]: SEQNUM=633
```

Un exemple pour connecter un disque dur :

```
mdev[41]: 7.267527 add@/devices/pci0000:00/0000:00:07.1/ata1/host0/target0:0:0/0:0:0/block/sda
mdev[41]: ACTION=add
mdev[41]: DEVPATH=/devices/pci0000:00/0000:00:07.1/ata1/host0/target0:0:0/0:0:0/block/sda
mdev[41]: SUBSYSTEM=block
mdev[41]: MAJOR=8
mdev[41]: MINOR=0
mdev[41]: DEVNAME=sda
mdev[41]: DEVTYPE=disk
mdev[41]: SEQNUM=688
```

Ci-dessus est un disque dur ATA/IDE (ata1), le nom du périphérique `sda` sera utilisé pour le traitement.

Un exemple pour un périphérique de blocage à distance (affecte le marquage d'un fichier d'image de VM-flr4l et a été résolu via `virsh detach-device`) :


```

mdev[42]: 52.600646 remove@/devices/pci0000:00/0000:00:0a.0/virtio5/block/vdb/vdb1
mdev[42]: ACTION=remove
mdev[42]: DEVPATH=/devices/pci0000:00/0000:00:0a.0/virtio5/block/vdb/vdb1
mdev[42]: SUBSYSTEM=block
mdev[42]: MAJOR=254
mdev[42]: MINOR=17
mdev[42]: DEVNAME=vdb1
mdev[42]: DEVTYP=partition
mdev[42]: SEQNUM=776

mdev[42]: 52.644642 remove@/devices/virtual/bdi/254:16
mdev[42]: ACTION=remove
mdev[42]: DEVPATH=/devices/virtual/bdi/254:16
mdev[42]: SUBSYSTEM=bdi
mdev[42]: SEQNUM=777

mdev[42]: 52.644718 remove@/devices/pci0000:00/0000:00:0a.0/virtio5/block/vdb
mdev[42]: ACTION=remove
mdev[42]: DEVPATH=/devices/pci0000:00/0000:00:0a.0/virtio5/block/vdb
mdev[42]: SUBSYSTEM=block
mdev[42]: MAJOR=254
mdev[42]: MINOR=16
mdev[42]: DEVNAME=vdb
mdev[42]: DEVTYP=disk
mdev[42]: SEQNUM=778

mdev[42]: 52.644777 remove@/devices/pci0000:00/0000:00:0a.0/virtio5
mdev[42]: ACTION=remove
mdev[42]: DEVPATH=/devices/pci0000:00/0000:00:0a.0/virtio5
mdev[42]: SUBSYSTEM=virtio
mdev[42]: MODALIAS=virtio:d00000002v00001AF4
mdev[42]: SEQNUM=779

mdev[42]: 52.644973 remove@/devices/pci0000:00/0000:00:0a.0
mdev[42]: ACTION=remove
mdev[42]: DEVPATH=/devices/pci0000:00/0000:00:0a.0
mdev[42]: SUBSYSTEM=pci
mdev[42]: PCI_CLASS=10000
mdev[42]: PCI_ID=1AF4:1001
mdev[42]: PCI_SUBSYS_ID=1AF4:0002
mdev[42]: PCI_SLOT_NAME=0000:00:0a.0
mdev[42]: MODALIAS=pci:v00001AF4d00001001sv00001AF4sd00000002bc01sc00i00
mdev[42]: SEQNUM=780

```

Comme vous pouvez le voir, plusieurs extraction du sous-système du kernel ont été impliqués (avec block, bdi, virtio et pci).

1.7.5 Périphériques ttyI

Au sujet les périphériques ttyI, vous pouvez utiliser (/dev/ttyI0 .../dev/ttyI15), pour la création d'un « émulateur de modem » avec plusieurs cartes ISDN (ou RNIS), il existe un compteur pour les conflits entre les différents OPTs et les utilisations de ces périphériques, c'est un dispositif à éviter. Ces émulateur seront créés lors du démarrage du routeur, dans le fichier /var/run/next_ttyI, avec l'utilisation un compteur. Dans l'exemple de script suivant, la valeur est interrogé et peut être augmentée de un, il sera exporté de nouveau dans le prochain OPT.

```

ttydev_error=
ttydev=$(cat /var/run/next_ttyI)
if [ $ttydev -le 16 ]
then
                                # ttyI device available? yes

```

```

ttydev=$((ttydev + 1))           # ttyI device + 1
echo $ttydev >/var/run/next_ttyI # save it
else                             # ttyI device available? no
    log_error "No ttyI device for <Nom de votre OPT> available!"
    ttydev_error=true            # set error for later use
fi

if [ -z "$ttydev_error" ]        # start OPT only if next tty device
then                             # was available to minimize error
    ...                          # messages and minimize the
                                # risk of uncomplete boot
fi

```

1.7.6 Scripts de connexion et de déconnexion par modem

Généralités

Après avoir établi ou coupé une connexion par modem, les scripts `/etc/ppp/` sont traitées. Voici les actions qui sont nécessaires pour activer ou désactiver une connexion, qui seront stocker dans l'OPT. Le schéma des noms de fichier est le suivant :

```

ip-up<numéro à trois chiffres>.<nom d'OPT>
ip-down<numéro à trois chiffres>.<nom d'OPT>

```

Le script `ip-up` est exécuté après la *connexion* et le script `ip-down` est exécuté après la *déconnexion*

Important: Dans le script `ip-down` aucune intervention ne doit être réalisé, autrement cela conduirait à une nouvelle connexion Internet, avec uniquement un accès à l'état online permanent, pour les utilisateurs qui non pas de forfait illimité, cela peut couter très cher.

Important: Car pour ce script, il n'y a aucun processus qui est généré, en plus, ce script ne peut pas être arrêté avec la commande « `exit` » !

Remarque : le scripts `ip-up` peut être examiné lors qu'il est exécuté, pour cela le fichier `rc400` sera vérifier avec la variable `ip_up_events`. Si c'est réglé sur "yes", une connexion par modem existe et le script `ip-up` sera exécuté. Si c'est réglé sur "no", une connexion par modem n'existe pas et le script `ip-up` ne sera pas exécuté. Il y a une exception pour cette règle : Si un routeur Ethernet pur n'est pas configuré pour des connexions commutées, mais configuré pour une route par défaut (0.0.0.0/0), le script `ip-up` sera exécuté qu'une fois, exactement à la fin du processus de boot. (De même le script `ip-down` sera exécuté qu'une fois avant l'arrêt du routeur).

Les variables

Grâce au concept d'appel spécial les scripts `ip-up` et `ip-down` sont exécutés et les variables suivantes sont utilisées :

Route par défaut

Depuis la version 2.1.0, les scripts `ip-up` et `ip-down` sont exécutés non seulement pour l'interface sur laquelle la route par défaut est configurée, mais aussi pour toutes les connexions

<code>real_interface</code>	L'interface actuelle, par ex. <code>ppp0</code> , <code>ippp0</code> , ...
<code>interface</code>	Interface-IMOND, avec <code>pppoe</code> , <code>ippp0</code> , ...
<code>tty</code>	Terminal de connexion, peut-être vide!
<code>speed</code>	Vitesse de connexion, par ex. avec ISDN 64000
<code>local</code>	Adresse-IP spécifique
<code>remote</code>	Adresse-IP d'ordinateur auquel vous êtes connecté
<code>is_default_route</code>	Indique si l'actuel <code>ip-up/ip-down</code> est utilisé pour l'interface de la route par défaut peut-être « yes » ou « no »)

qui ont besoin les scripts `ip-up` et `ip-down`. Pour émuler des comportements anciens, vous devez inclure les éléments à déclencher dans les scripts `ip-up` et `ip-down` la requête suivante doit être insérée :

```
# is a default-route-interface going up?
if [ "$is_default_route" = "yes" ]
then
    # Les actions à déclencher
fi
```

Naturellement, les nouveaux comportements doivent être utilisés, pour des actions spécifiques.

1.8 Paquetage Template

Pour illustrer quelques-uns des objectifs décrits ci-dessus, vous avez un paquetage avec des modèles pour la distribution `fli4l`. Dans ce paquetage vous avez une série de petits exemples, tels que :

- Voir un fichier config dans (`config/template.txt`)
- Un fichier de contrôle qui est écrit dans (`check/template.txt`)
- L'extension des fonctions de contrôle dans (`check/template.ext`)
- Des variables de configuration pour une utilisation ultérieure stockés dans (`opt/etc/rc.d/rc999.template`)
- Des variables de configuration pour être lu à nouveau stockés dans (`opt/usr/bin/template_show_config`)

1.9 Construction du Boot sur un support de données

Depuis de la version 1.5 `fli4l` utilise le programme `syslinux` pour booter. Il a l'avantage d'avoir un système de fichier DOS compatible sur le support de données.

Le support de données pour le boot contient les fichiers suivants :

Au démarrage du script `mkfli4l.sh` (ou `mkfli4l.bat` pour DOS) les fichiers `opt.img`, `syslinux.cfg`, `rc.cfg`, ainsi que `rootfs.img` sont d'abord exécutés. Le programme `mkfli4l` exécute les fichiers nécessaires des répertoires `unix` ou `windows` pour l'installation. Dans les deux archives, le Kernel et les autres paquetages à installer sont inclus. Le fichier `rc.cfg` se trouve à la fois dans l'archive `opt` et sur le boot disque (ou disque de démarrage).¹²

12. Le contenu du fichier dans l'archive `opt` est nécessaire au début de la phase de boot, car à ce moment là le volume de boot n'est pas monté.

<code>ldlinux.sys</code>	Le chargeur (« Boot loader ») <code>syslinux</code>
<code>syslinux.cfg</code>	Fichier de configuration pour <code>syslinux</code>
<code>kernel</code>	Linux-Kernel
<code>rootfs.img</code>	RootFS : contient les programmes nécessaires pour le Boot
<code>opt.img</code>	Fichier Optionnel : drivers et Opt-Paquetage
<code>rc.cfg</code>	Fichier de configuration des variables utilisées depuis le répertoire <code>config</code>
<code>boot.msg</code>	Texte pour le menu de démarrage <code>syslinux</code>
<code>boot_s.msg</code>	Texte pour le menu de démarrage <code>syslinux</code>
<code>boot_z.msg</code>	Texte pour le menu de démarrage <code>syslinux</code>
<code>hd.cfg</code>	Fichier de configuration pour l'attribution des partitions

Ensuite, l'ensemble des fichiers du Kernel, `rootfs.img`, `opt.img` et `rc.cfg`, sont copiés avec le fichier `syslinux` sur le support de données.

Lors du boot de `fl4l` le script `rc.cfg` dans `/etc/rc` est analysé et l'archive `opt.img` compressée est intégrée à la racine du système de fichiers du disque virtuel (selon le type d'installation, les fichiers seront décompressés directement à la racine du système de fichiers du disque virtuel ou vers le lien symbolique inclus). Pour terminer, les scripts dans le répertoire `/etc/rc.d` sont exécutés dans l'ordre alphanumérique, ensuite les pilotes sont chargés et les services démarrent.

1.10 Fichiers de configurations

Voici les dossiers présélectionnés par le routeur `fl4l` on-the-fly, qui sont générés au moment du boot.

1. Configuration du fournisseur
 - `etc/ppp/pap-secrets`
 - `etc/ppp/chap-secrets`
2. Configuration du DNS
 - `etc/resolv.conf`
 - `etc/dnsmasq.conf`
 - `etc/dnsmasq-dhcp.conf`
 - `tc/resolv.dnsmasq`
3. Fichier hôte
 - `etc/hosts`
4. Configuration de `imond`
 - `etc/imond.conf`

1.10.1 Configuration du fournisseur

Pour paramétrer l'ID de l'utilisateur et le mot de passe pour le fournisseur d'accès cela doit se faire dans le fichier `etc/ppp/pap-secrets`.

Exemple pour le fournisseur Planet-Interkom :

```
# Secrets pour l'authentification en utilisant PAP
# client      server      secret      IP adresse
"anonymmer"   *          "surfer"    *
```

Dans cet exemple, l’ID de l’utilisateur est « anonym ». l’accès au serveur distant sera permis à tous (avec « * »). « surfer » est le mot de passe pour le fournisseur Planet-Interkom.

1.10.2 Configuration DNS

Vous pouvez utiliser le routeur fli4l comme un serveur DNS. Pourquoi cette fonction est utile et même obligatoire dans un LAN avec des ordinateurs Windows ? Tout est expliqué dans la documentation du paquetage « base ».

Le fichier résolveur `etc/resolv.conf` contient les noms de domaine et les utilisateurs du serveur de nom. Vous pouvez voir ci-dessous le contenu (du « domain.de », vous avez seulement un espace réservé pour indiquer les paramètres c’est dans la variable de configuration `DOMAIN_NAME`) :

```
search domain.de
nameserver 127.0.0.1
```

Le serveur de noms dnsmasq est configuré en utilisant le fichier `etc/dnsmasq.conf`. Il boot à partir du script `rc040.dnsmasq` et du `rc370.dnsmasq` qui sont générés automatiquement cela pourrait ressembler à ceci :

```
user=dns
group=dns
resolv-file=/etc/resolv.dnsmasq
no-poll
no-negcache
bogus-priv
log-queries
domain-suffix=lan.fli4l
local=/lan.fli4l/
domain-needed
expand-hosts
filterwin2k
conf-file=/etc/dnsmasq_dhcp.conf
```

1.10.3 Fichier hôte

Ce fichier contient l’ensemble des noms d’hôtes avec leurs adresses IP. Donc, ce classement est applicable seulement pour un fli4l local, pour les autres ordinateurs dans le LAN, il ne sera pas visible. Ce fichier n’est pas vraiment nécessaire, si un serveur DNS local est déjà démarré.

1.10.4 Configuration de imond

Les variables de configurations pour le fichier `etc/imond.conf` doivent entre autre être activées, `CIRC_x_NAME`, `CIRC_x_ROUTE`, `CIRC_x_CHARGEINT` et `CIRC_x_TIMES`. Il peut être constituée d’un maximum de 32 lignes (à l’exclusion des lignes de commentaires). Chaque ligne est composée de 8 colonnes :

1. Plage journalière
2. Plage horraire
3. Dispositif (`ippXX` ou `isdnX`)

4. Circuit pour Default-Route : « yes »/« no »
5. Numéro de Téléphone
6. Nom du Circuit
7. Prix de l'unité Téléphonique par Minute en EU
8. Compteur (interval d'unité-Tél) en seconde

Voici un exemple :

#day	hour	device	defroute	phone	name	charge	ch-int
Mo-Fr	18-09	ipp0	yes	010280192306	Addcom	0.0248	60
Sa-Su	00-24	ipp0	yes	010280192306	Addcom	0.0248	60
Mo-Fr	09-18	ipp1	yes	019160	Compuserve	0.019	180
Mo-Fr	09-18	isd2	no	0221xxxxxxx	Firma	0.08	90
Mo-Fr	18-09	isd2	no	0221xxxxxxx	Firma	0.03	90
Sa-Su	00-24	isd2	no	0221xxxxxxx	Firma	0.03	90

D'autres explications peuvent être trouvées pour Least-Cost-Routing (ou routage au moindre coût) dans la documentation du paquetage « base ».

1.10.5 Le fichier `/etc/.profile`

Le fichier `/etc/.profile` contient les paramètres par défaut du Shell. Pour modifier le fichier `/etc/.profile` par défaut, il est nécessaire d'ajouter les paramètres en dessous des paramètres existant. Ces paramètres peuvent être utilisés pour paramétrer un raccourci d'une commande Prompt et ensuite (vous pourrez exécuter « ce raccourci »).

Important: *Ce fichier ne doit pas contenir le paramètre `exit` !*

Exemple :

```
alias ll='ls -al'
```

1.10.6 Les scripts dans `/etc/profile.d/`

Vous pouvez stocker un script dans le répertoire `/etc/profile.d/`, ce script sera exécuté au démarrage du shell et ainsi influencer l'environnement du shell. Généralement, les développeurs de programme OPT, y placent des scripts qui définissent des variables d'environnement spéciales nécessaires au programme OPT.

Si des scripts sont situés dans le répertoire `/etc/profile.d/` et s'il existe un fichier script `/etc/.profile`, les scripts du répertoire `/etc/profile.d/` seront exécutés après le fichier script `/etc/.profile`.

Table des figures

Liste des tableaux

1.1	Paramètres pour <code>mkfli41</code>	6
1.2	Options pour les fichiers	9
1.3	Expressions logiques	31
1.4	Les fonctions pour le script <code>cgi-helper</code>	45

Index

DEBUG_ENABLE_CORE, [38](#)
DEBUG_IP, [38](#)
DEBUG_IPUP, [38](#)
DEBUG_KEEP_BOOTLOGD, [38](#)
DEBUG_MDEV, [39](#)

LOG_BOOT_SEQ, [38](#)