Faciliter la configuration d'un logiciel avec Config::Model

Dominique Dumont

Hewlett-Packard

French Perl Workshop, Paris 2009

Outline

La config, c'est souvent pénible !

Pour un utilisateur, éditer la configuration est souvent difficile :

- Edition d'un fichier texte en dehors de /home
- Lecture de pages de manuel
- Assurer la cohérence
- Encore plus délicat lors des mises-à-jour de paquets

Objectif 1 : faciliter la configuration pour les utilisateurs

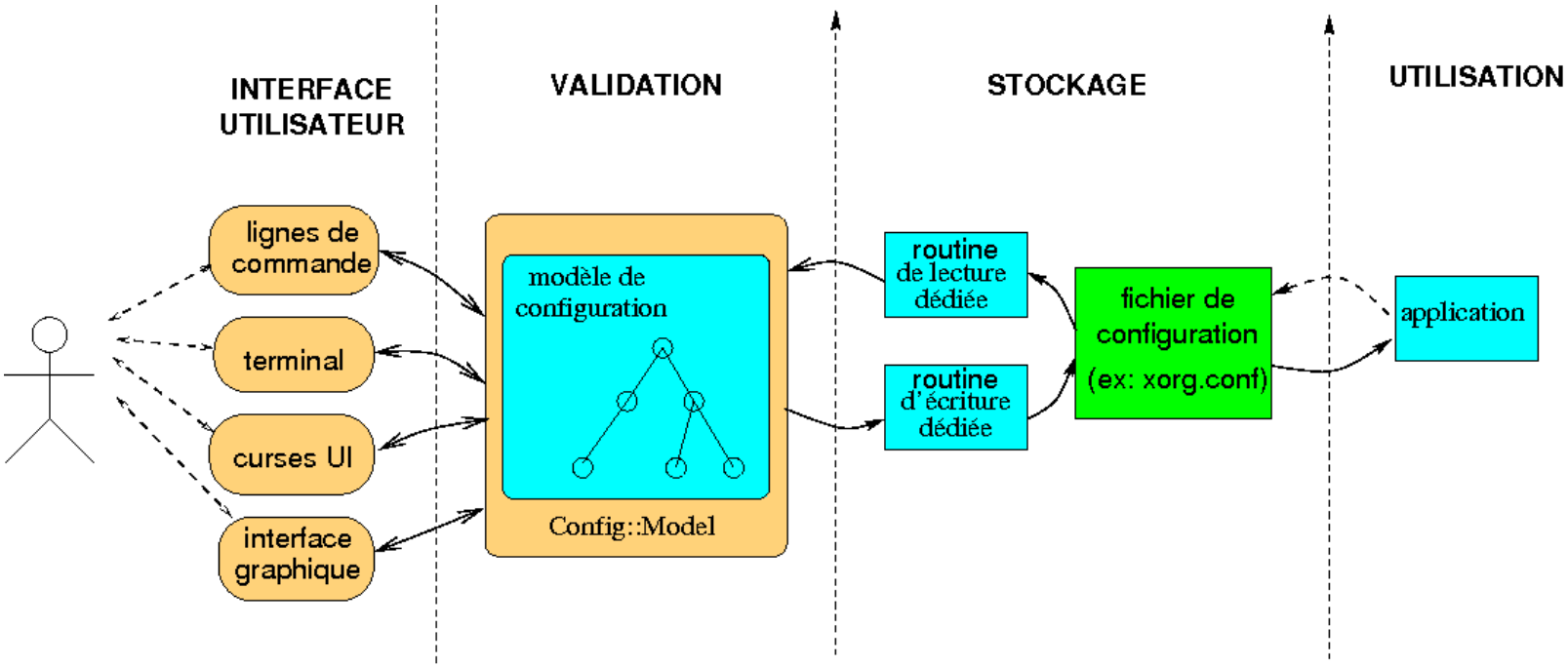
Fournir une interface graphique avec :

- Aide intégrée
- Rappel des valeurs par défaut
- Validation des données de configuration
- Gestion des mises-à-jour (config upgrade)
- Différents niveaux d'expertise *(du débutant au grand maître)*
- Recherche
- Tout ceci existe plus ou moins :
  - Webmin
  - Windows registry
  - Elektra

Objectif 2 : ne pas compliquer la vie des développeurs

- La logique de validation des données de config soit rester facile à maintenir :
  - Éviter le code dédié à la validation des données (e.g. ne pas refaire un Webmin)
  - Baser la validation sur des «méta-données» : le modèle de configuration
  - Générer les interfaces (graphiques ou pas) à partir du modèle
  - Faciliter les mises-à-jour de la configuration des utilisateurs
- Minimiser le code requis pour lire ou écrire les fichiers de configuration :
  - Utiliser les librairies existantes (Config : :Ini, Config : :Augeas...)
  - Fournir des classes pour aider à la lecture et écriture

Schéma de Config::Model



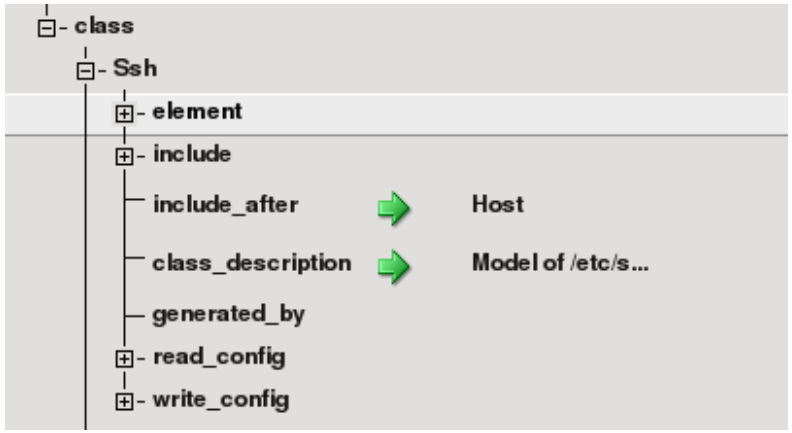
C'est quoi un modèle ?

Un modèle définit une structure en arbre :

- Chaque nœud est une classe
- Chaque feuille est un paramètre

Chaque classe contient :

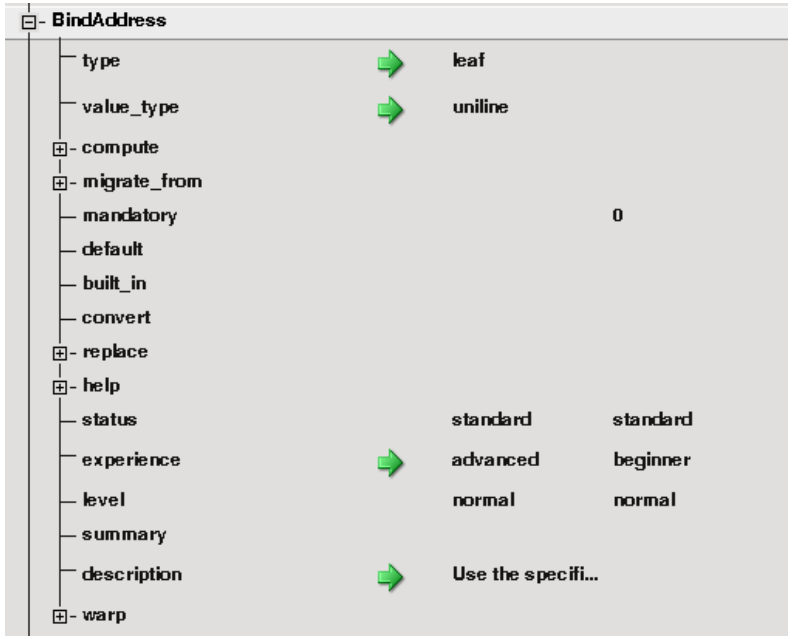
- un ensemble d'éléments (les paramètres)
- en option : une spécification pour l'accès au fichier de conf (*backend*)



Un élément

Chaque élément a :

- un type (*leaf, hash, list, node*)
- des contraintes (*entier, max, mini* ...)
- une valeur par défaut
- une description et un résumé (pour l'aide en ligne)
- un niveau d'expérience (*beginner, advanced master*)
- un statut (normal ou obsolète)



Élément complexe

Un modèle de configuration peut aussi définir des interactions entre les éléments :

- Déformation du modèle (exemple : avec Xorg, les options d'un pilote changent en fonction du pilote déclaré) (*warp*)
- Calcul simples à partir d'autres paramètres (utiles pour les mises-à-jour) (*compute* et *migrate_from*)
- Des références (exemple : dans Xorg : :Device : :Radeon, Monitor-DVI-0 doit choisir un des moniteurs déclarés dans la section Monitor)

Modèle : analyse

- Lire les pages de manuel de l'application :
  - Trouver la structure
  - Identifier les paramètres, leur contraintes et relations
- Trouver plusieurs exemples valides :
  - Pour vérifier qu'on a compris la documentation
  - Pour les tests de non-régression

Modèle : déclaration

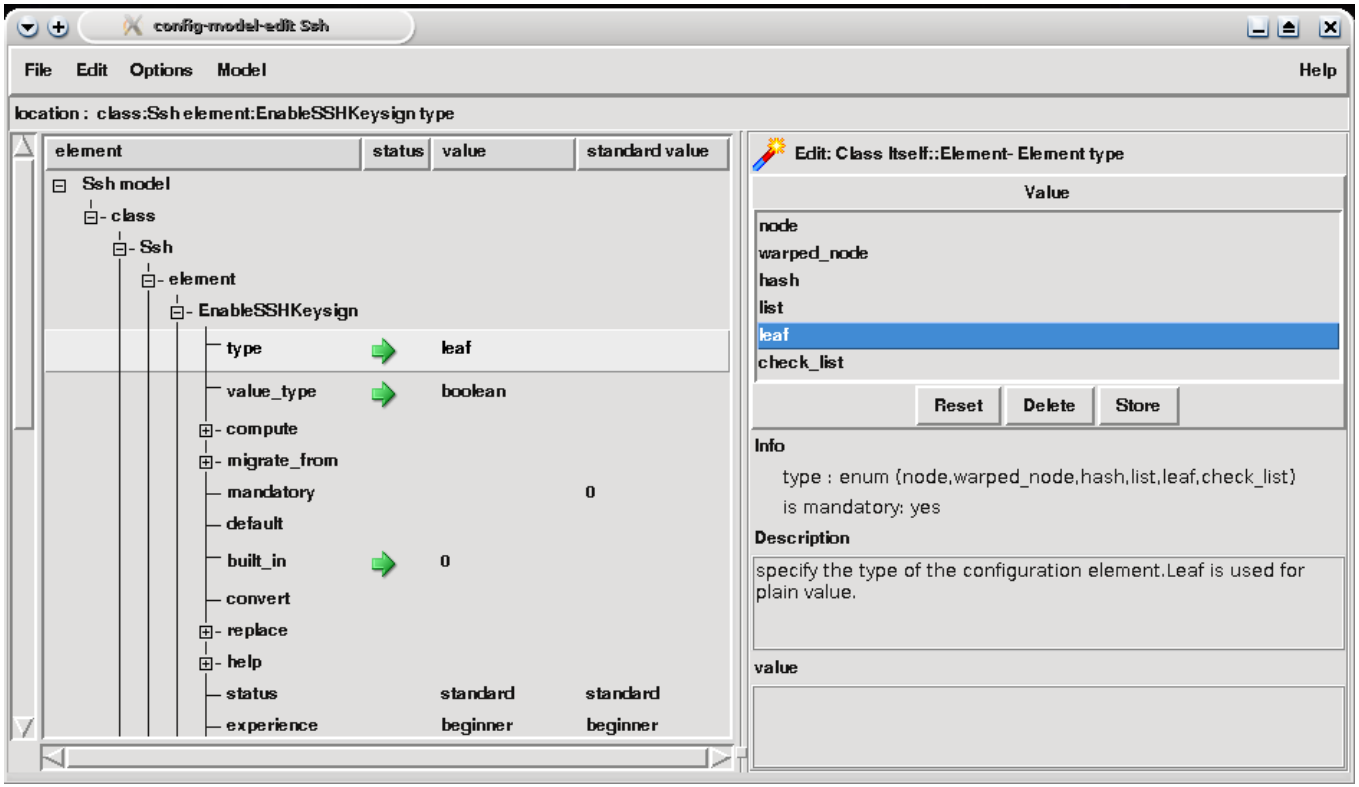
En gros, on traduit la documentation en un format compréhensible par Config::Model :

- La structure va se traduire en *classes de configuration*
- Les paramètres de configuration en éléments
- Les contraintes en attributs des éléments

```
name => 'Ssh',                  # class name
element => [
  EnableSSHKeysign => {         # element name
    type => 'leaf',
    value_type => 'boolean',
    built_in => '0',            # default value
    description => 'Setting ...',
  },
]
```

Déclaration (en plus facile)

Comme écrire une structure de donnée n'est pas drôle (même en Perl). On peut créer le modèle avec une GUI :



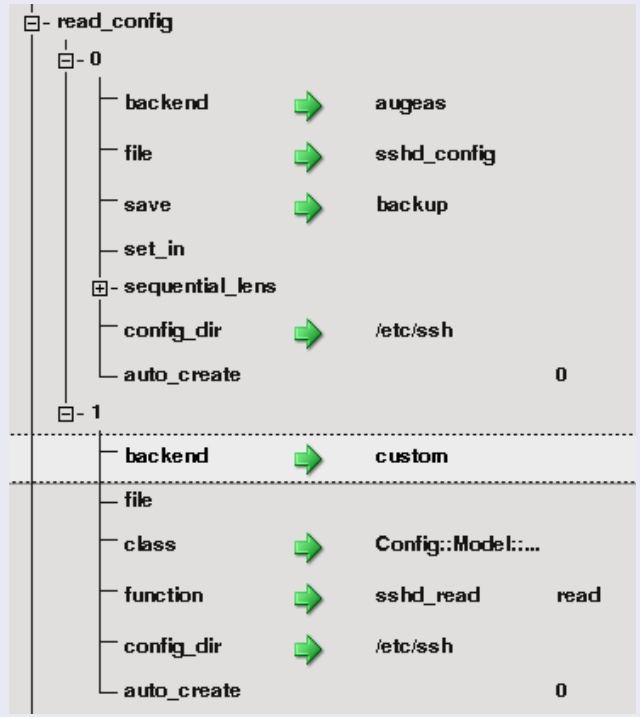
Ne pas oublier, de temps en temps, Menu → Model → test ☺

Comment lire les fichiers

Dans le modèle

- Déclarer la méthode (*backend*)
  - Fournie (fichier Perl, Ini...)
  - Plug-in (classe *Backend*)
  - Spécifique (*custom* → call-back à écrire)
- Les paramètres
- Les spécifications sont essayées dans l'ordre

Exemple



Comment écrire les fichiers

Dans le modèle

- Inutile si la spécification est identique à celle de la lecture
- Même paramètres que pour la lecture
- Essayées dans l'ordre jusqu'au premier succès

A noter

Ces spécifications permettent de migrer une configuration d'une syntaxe à une autre

Exemple

```
write_config => [
 {
  backend     => 'augeas',
  save        => 'backup',
  config_dir  => '/etc/ssh',
  file        => 'sshd_config',
 },
 {
  backend    => 'custom',
  class      => 'C : :M : :OpenSsh',
  function   => 'sshd_write',
  config_dir => '/etc/ssh'
 }
],
```

Prévoir pour les mises-à-jour des config

Pour des évolutions transparentes

Pour les concepteurs d'application :

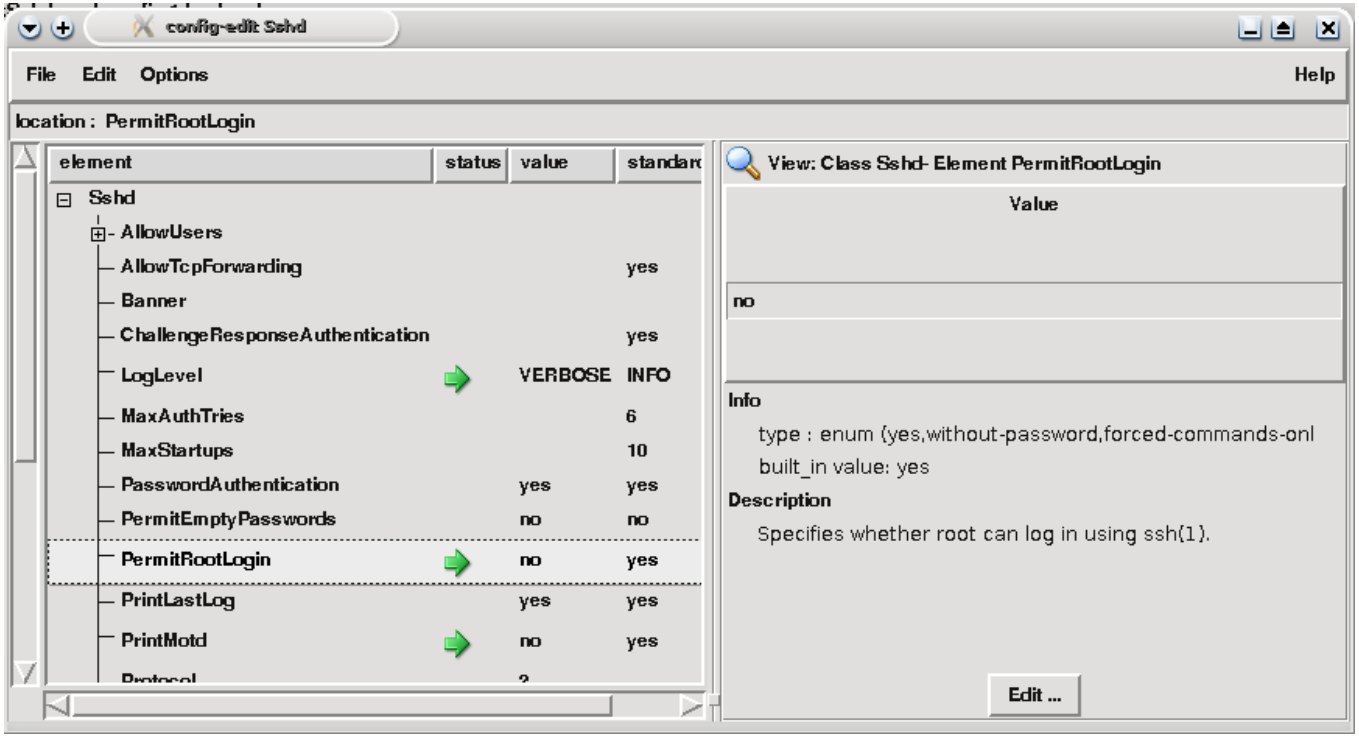
1. Pas de nouveaux paramètres $\leftrightarrow$ pas de nouveaux problèmes
2. Choix des noms et valeurs de paramètres : Un bon nom vaut mieux que 3 pages de doc
3. Valeur par défaut : Fonctionne même avec un fichier vide

Mais, s'il le faut, le modèle peut spécifier :

- comment remplacer une valeur *replace*
- des paramètres obsolètes *status*
- comment migrer une valeur *migrate* avec une formule de calcul

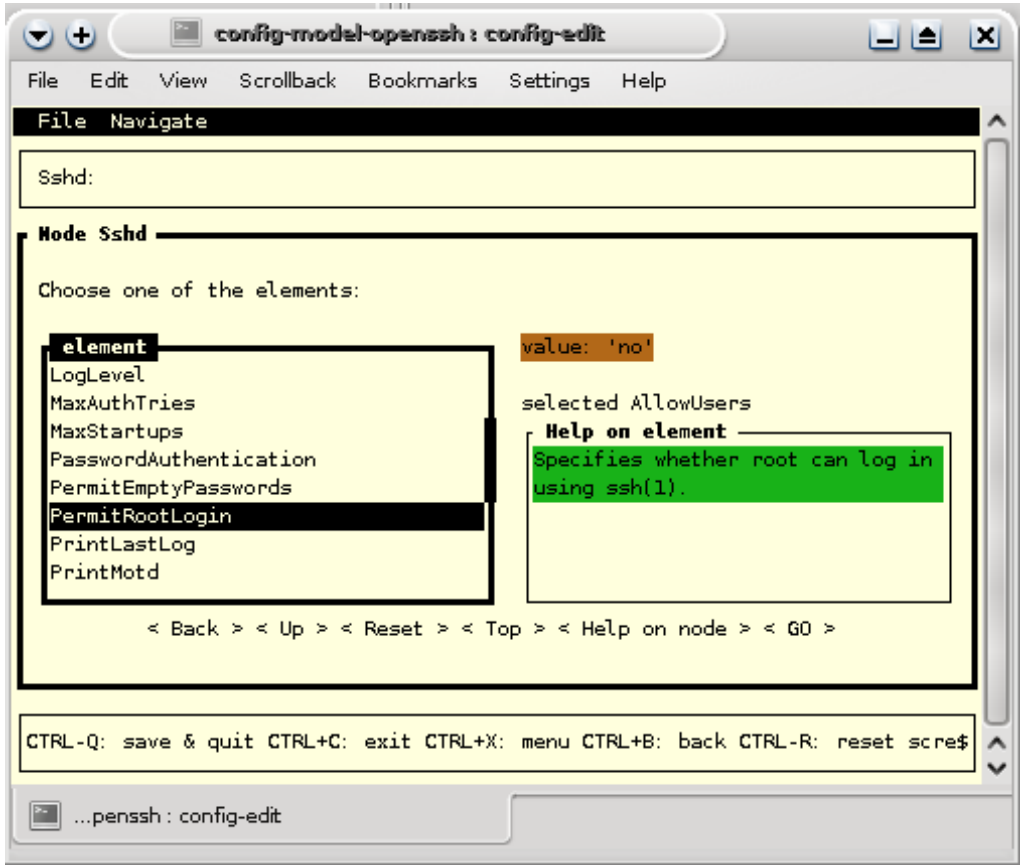
Pour plus d'informations sur les migrations appliquées aux paquets, voir `http://wiki.debian.org/PackageConfigUpgrade`

Utilisation
interface graphique

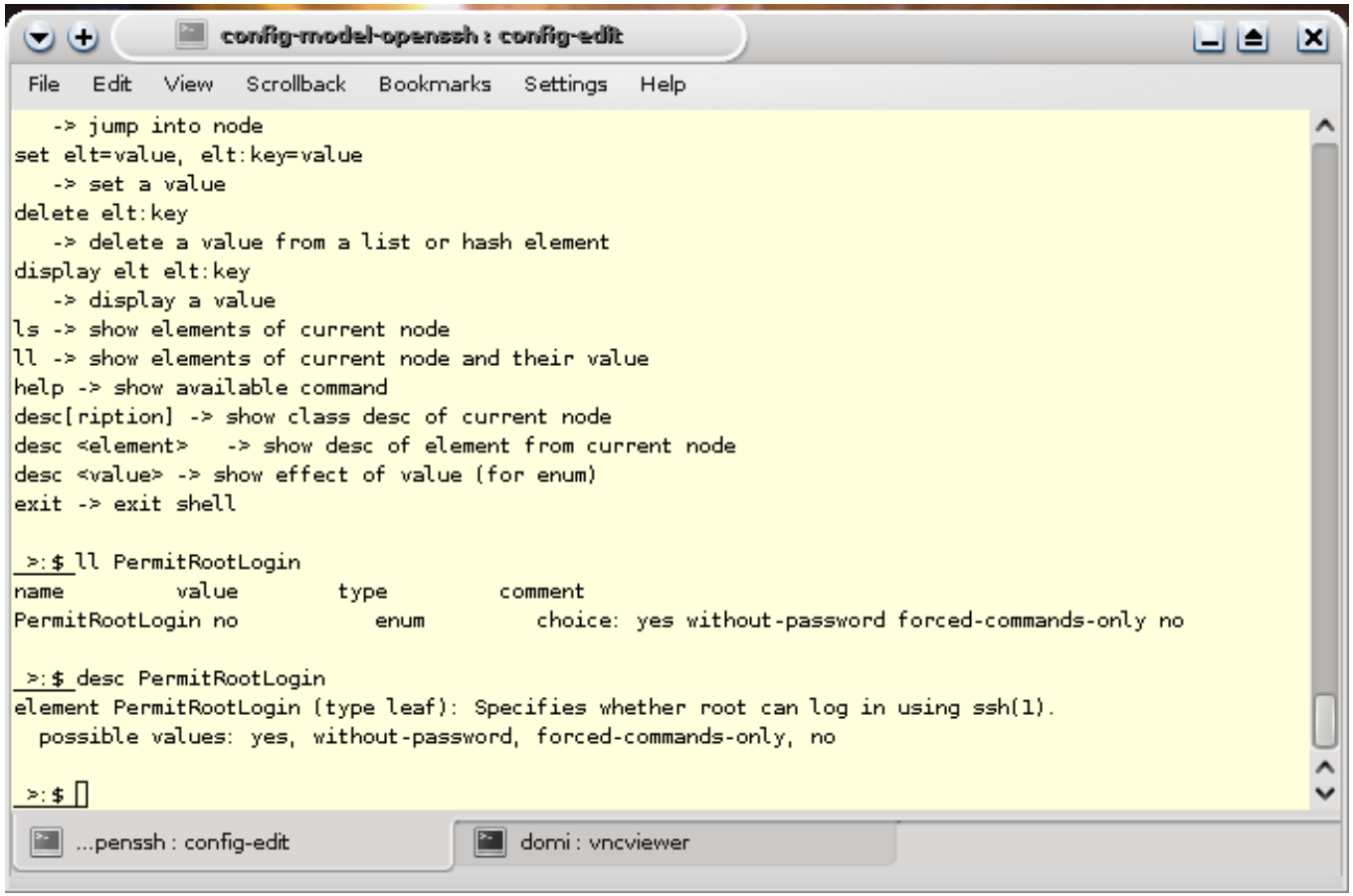


Note : Menu *Option* → *experience* pour révéler plus de paramètres

Utilisation
interface curses



Utilisation
interface shell



Utilisation

Pour les programmes

En ligne de commande :

```
$ sudo config-edit-sshd -ui none PermitRootLogin=no
2009/05/15 14 :03 :28 load model Config/Model/models/Sshd.pl
2009/05/15 14 :03 :28 Creating class Sshd
2009/05/15 14 :03 :29 Backing up file /etc/ssh/sshd_config
2009/05/15 14 :03 :29 writing config file /etc/ssh/sshd_config
```

En Perl :

```
$ sudo perl -MConfig : :Model -e '
  my $i = Config : :Model -> new -> instance(root_class_name=>"Sshd") ;
  $i->config_root->load("PermitRootLogin=no") ;
  $i->write_back ;'
```

État du projet

Modèles dispos

- OpenSsh
- Approx
- Config::Model
- `Xorg`

Backend

- syntaxe INI
- Structure de donnée en Perl
- Augeas

Communauté

- Paquets Debian
- Paquets Rpm en préparation
- Proposition pour *dh_config* (mise-à-jour paquets)
- article dans GNULinux Mag

Les futurs projets

Interfaces

- *Wizard* pour la GUI
- Recherche de paramètres et valeurs
- Annotations

backend

- JSON
- YAML
- XML

Et vous ?

Le projet Config::Model a besoin de votre aide :

- Intégration dans les distros
- Configuration multi-niveaux
- Mécanisme de plug-in pour les modèles (pilotes Xorg)
- Mécanisme à définir pour les injections de config (mercurial viewer dans apache)

Liens

- Site Config::Model
  `http://config-model.wiki.sourceforge.net`
- Config::Model sur CPAN
  `http://search.cpan.org/dist/Config-Model/`
- Liste des utilisateurs de Config::Model `https://lists.sourceforge.net/lists/listinfo/config-model-users`
- GNU/Linux Mag n°117 « Config::Model - Créer un éditeur graphique de configuration avec Perl (1ère partie)»
- Proposition pour l'integration de Config::Model pour gérer les configuration pendant la mise à jour des paquets Debian
  `http://wiki.debian.org/PackageConfigUpgrade`
- Projet Augeas `http://augeas.net`