Licence d'informatique
Module de C/Unix

Les commandes `pod2*`

Examen – Partie Unix

Philippe MARQUET

Septembre 2003

Les documents de cours et TD sont autorisés.
On rendra deux copies séparées pour la partie langage C et pour la partie Unix ; la partie Unix sur une *copie de couleur*.
On remarquera que les questions peuvent être traitées indépendamment les unes des autres.
On introduira tout code Bourne-shell par une brève description de son comportement ; on attachera un commentaire à toute portion non triviale de code.

1 Questions de cours

Exercice 1 (`ftp` *vs.* `rcp`)
Les commandes `ftp` et `rcp` permettent de transférer des fichiers entre deux machines. Expliquez les avantages respectifs de ces deux commandes. □

Exercice 2 (`cp` *vs.* `mv`)
Certains environnements graphiques de manipulation de fichiers (*file manager*) ont le comportement suivant quand on fait *glisser* un fichier vers un répertoire :
- si le fichier et le répertoire sont sur le même disque et que les propriétaires du fichier et du répertoire sont les mêmes, la commande `mv` est utilisée ;
- sinon, la commande `cp` est utilisée.

Expliquez ce qui peut justifier un tel comportement. □

Exercice 3 (Renommer les `.sh`)
Expliquez pourquoi une commande `rename` qui voudrait renommer l'ensemble de fichiers `.sh` en `.bash` ne peut être appelée depuis le shell par la séquence :

```
% rename *.sh *.bash
```

□

2 Documentons notre code, documentez votre code[1]

Les commandes Unix sont documentées. La commande `man` présente la page de manuel d'une commande donnée. Cette page de manuel est organisée en sections (`NAME`, `SYNOPSIS`, `DESCRIPTION`, `OPTIONS`...). Voir le cours. En particulier la section `SYNOPSIS` donne la syntaxe et l'usage de la commande.

Le problème des commandes développées par les utilisateurs, par exemple sous forme de scripts, est que ces commandes sont souvent non documentées (il en est ainsi par exemple pour les commandes développées par un étudiant dans le cadre d'un projet d'Unix...).

L'objet des commandes développées ici est de faciliter la production de cette documentation.

[1] Merci à Jeffreys Copeland & Haemer pour leur chronique de décembre 2001 dans Server/Workstation Expert.

Id: xam03-sep.tex,v 1.5 2003/09/14 16:23:02 marquet Exp

2.1 Le format POD

Le format POD est un format de documentation défini par Larry Wall, le concepteur de Perl. Un des avantages majeurs de POD est de pouvoir intégrer la documentation dans le code même de la commande documentée. Le but étant de fournir aux programmeurs un moyen simple d'écrire de la documentation, espérant ainsi qu'ils écriront cette documentation !

Nous nous limiterons à la syntaxe suivante du format POD :
- une commande POD commence par un signe égal (=), suivi d'un identificateur, puis un éventuel paramètre ;
- un paragraphe indenté par au moins un espace ou une tabulation. Le texte de ce paragraphe sera reproduit tel-quel. Deux paragraphes sont séparées par au moins une ligne vide.

Nous considérons les commandes POD suivantes :

`=pod` Début de code POD.

`=cut` Fin de code POD.

`=head1` ***`Texte`*** Construit un titre de niveau 1. Ce sont les sections des pages de manuel.

`=item` Introduit un nouvel élément de liste.

2.2 Un exemple

La figure 1 présente le source d'un « hello world » d'un script shell dont la documentation est intégrée en utilisant POD.

Les commandes `pod2text`, `pod2usage` et `podselect` vont lire le source de ce script pour en extraire la documentation ou une de ses parties :

```
% ./hello.sh --help
hello [--help] [--man] [greetee]
 greetee
   The world to whom salutations should be offered.
 --help
   Print more details about the arguments.
 --man
   Print a full man page.
%
```

2.3 Les commandes `pod2*`

La commande `pod2text` transforme du source POD en un texte mis en page pour une lecture simple. Ce source POD est lu sur l'entrée standard ou dans le fichier désigné par un argument.

Les deux commandes `pod2usage` et `podselect` acceptent une syntaxe et un certain nombre d'options communes. La syntaxe est la suivante :

```
pod2usage [--msg message] [--file] [--pathlist dirlist] command...
podselect [-s section]... [--msg message] [--file]
          [--pathlist dirlist] command...
```

Ces deux commandes vont traiter un ou plusieurs fichiers sources dans lesquels se trouve de la documentation au format POD. Chacun de ces fichiers sources est identifié par un argument de type *`command`*. Le plus souvent, cet argument correspond au paramètre `$0` du script depuis lequel `pod2usage` ou `podselect` est appelé. Le fichier source est identifié de la manière suivante :
- si l'argument *`command`* correspond à un nom de fichier existant, ce fichier est utilisé ;
- sinon, un fichier de nom *`command`* est recherché dans les répertoires listés par la variable `$PATH`.

```
#! /bin/sh

usage() {
    pod2usage --msg "$0 --help for more information" $0
    exit 1
}

for option in "$@" ; do
    case "$option" in
        --man)
            pod2text $0
            exit 0 ;;
        --help)
            pod2usage $0
            podselect -s 'OPTIONS AND ARGUMENTS' $0 | pod2text
            exit 0 ;;
        -*) usage ;;
    esac
done

echo hello, ${1:-world}

exit 0

=pod
=head1 NAME

 hello - hello, world

=head1 SYNOPSIS

 hello [--help] [--man] [greetee]

=head1 DESCRIPTION

 The first program to write is the same for all languages:
 Print the words "hello, world"

=head1 OPTIONS AND ARGUMENTS

=item greetee

 The world to whom salutations should be offered.

=item --help

 Print more details about the arguments.

=item --man

 Print a full man page.

=head1 AUTHOR

 Jeffrey Copeland <copeland@alumni.caltech.edu>
 Jeffrey S. Haemer <jsh@usenix.org>

=cut
```

FIG. 1 – `hello.sh` : Hello world POD

La commande `podselect` affiche les sections sélectionnées de la documentation POD sur la sortie standard. L'option `-s` *`section`* identifie les sections devant être affichées. Cette options peut être répétée autant de fois que nécessaire sur la ligne de commande. L'argument *`section`* correspond au paramètre de la commande POD `=head1`. Typiquement, le résultat de `podselect` doit être filtré par `pod2text`.

La commande `pod2usage` affiche sur la sortie standard le texte correspondant à la documentation POD de la section `SYNOPSIS`.

La signification des options communes est la suivante :

`--msg` ***`message`*** Affiche le texte du message avant de produire le résultat de la commande.

`--file` Considère l'argument *`command`* uniquement comme un nom de fichier et le recherche ainsi pas dans les répertoires listés par `$PATH`.

`--pathlist` ***`dirlist`*** Si le fichier *`command`* n'existe pas, il est recherché dans les répertoires listés par *`dirlist`* et non dans `$PATH`. Cette liste est au format standard de `$PATH` : les éléments sont séparés par le caractère deux points **`:`**.

En cas d'erreur, commande *`command`* non trouvée, source POD absent dans le fichier... un message d'erreur est produit et une valeur adéquate retournée.

2.4 Questions

Exercice 4
Expliquez le pourquoi du dernier `exit 0` dans le code de la figure 1 alors que le code `=pod` délimite le début de code POD et donc la fin du script. □

Exercice 5

1. Expliquez la manière dont est délimité le texte d'une section dans le source POD d'un fichier.
2. Donnez le code d'une fonction Bourne-shell `select_section` qui lit sur l'entrée standard un fichier texte et affiche sur la sortie standard le source POD de la section dont le nom est passé en paramètre.

□

Exercice 6
Donnez le code d'une fonction Bourne-shell `command_to_file` qui retourne le nom de fichier correspondant à un paramètre *`command`* de la ligne de commande passé en paramètre à la fonction. Le résultat est fonction des variables `$FILE` et `$PATHLIST` qui ont été positionnées en fonction des options `--`*`file`* et `--`*`pathlist`* de la commande. □

Exercice 7
Donnez le code Bourne-shell de la commande `pod2usage`. □

Fin du sujet