Annales/UPMC/LI362/3

Environnement de développement

Annales corrigées novembre 2004-janvier 2006

Valérie Ménissier-Morain
Christian Queinnec
Guénaël Renault

PARACAMPLUS

Déjà paru :

UPMC/LI101 : annales corrigées

UPMC/LI102 : compléments de cours

Prochainement(?) :

UPMC/LI102 : annales corrigées

Missions de Paracamplus :

Paracamplus a pour vocation de procurer, à tous les étudiants en université, des ouvrages (annales corrigées, compléments de cours) et des environnements informatiques de travail, bref, des ressources pédagogiques propres à permettre, accélérer et entretenir l'apprentissage des connaissances enseignées.

Les ouvrages des éditions Paracamplus ont un format défini pour tenir commodément dans une poche et vous accompagner pendant vos déplacements en transports en commun. Ils bénéficient surtout d'une conception nous permettant de vous les proposer à des coûts très abordables.

Rafraîchir les annales tous les ans, procurer les compléments appropriés à chaque cours, vous proposer les meilleurs outils pour apprendre, tels sont les buts de ces ouvrages.

Découvrez-nous plus avant sur notre site **www.paracamplus.com**.

SARL Paracamplus

7, rue Viollet-le-Duc, 75009 Paris – France

ISBN 2-916466-02-9

Table des matières

Chapitre 1

Introduction

Cet ouvrage est lié à l'enseignement « Environnement de développement » dispensé en troisième année de licence d'informatique à l'université Pierre et Marie Curie (UPMC). Cet enseignement, aussi connu sous son code : LI362, ou son surnom : *envdev*, a été créé en octobre 2004, lors de l'introduction du LMD à l'UPMC, afin de faire en sorte que les étudiants de licence soient à l'aise sur des ordinateurs.

Tout informaticien en effet se doit d'apprendre à automatiser les tâches répétitives qu'il rencontre dans son travail. Cette automatisation passe par l'écriture d'un programme approprié dans un langage adapté. C'est le but de cet enseignement que de présenter les bases des techniques de développement à savoir les langages de commandes tels que **bash**, l'édition de texte avec **emacs**, la compilation et l'édition de liens (la connaissance du langage C est requise), **make** et l'écriture de *Makefile*, la mise au point avec les outils **gdb** et **ddd**, les tests unitaires, la gestion de versions **rcs/cvs**, sans oublier la compréhension de l'environnement de travail avec X, NFS et **ssh**.

Cet enseignement est décrit plus amplement sur le site de la licence d'informatique en `www.licence.info.upmc.fr`. Vous y trouverez notamment un certain nombre de ressources pédagogiques telles que les transparents de cours, des liens vers des documentations intéressantes, les bandes-sons de certains cours d'amphithéâtre ainsi que le matériel propre aux examens : les énoncés et les fichiers solutions.

1.1 Examens et ordinateurs

Une caractéristique peu commune de cet enseignement est que tous les examens ont lieu sur ordinateur : il s'agit de rendre des fichiers (de données ou de programmes) répondant à des spécifications précises tout en disposant de tous les outils ou documentations usuels. On n'imagine pas une épreuve de dessin où la seule épreuve consisterait à

disserter sur la façon de tenir un crayon ! Pourquoi, en programmation, ne demanderait-on que de disserter sur ce que l'on pourrait écrire si l'on avait à programmer telle ou telle chose ?

Une seconde caractéristique, plus rare et encore plus importante, est que l'examen final est corrigé automatiquement par programmes. Cette situation est professionnalisante en ce sens qu'elle est réaliste : il s'agit de remplir une tâche, précisément spécifiée, avec le langage de son choix, l'algorithmique de son choix (elle est toujours sans difficulté dans ce cours), la structuration de données de son choix. La notation automatique [QC02] opère de manière complètement aveugle et vérifie seulement que les propriétés attendues sont bien présentes. Cet aveuglement correspond bien aux procédures usuelles de test unitaire que l'on rencontre professionnellement : tout est libre à condition de respecter la spécification. La sanction est en revanche professionnelle : un seul point-virgule oublié, un nom de fichier mal écrit et rien ne marche plus ! Mais les ordinateurs sont justement employés pour détecter ces problèmes.

Dans le cadre du cours, les copies corrigées automatiquement sont également relues par les enseignants afin de déterminer les erreurs les plus courantes, les mauvais réflexes, les incompréhensions les plus communes. Ces annales rassemblent non seulement les solutions ou les variantes de solutions pour tous les partiels, examens et rattrapages, depuis novembre 2004 jusqu'à janvier 2006, mais commentent également ces solutions et notamment illustrent ce qu'il fallait faire ou ne pas faire.

1.2 Structure

Ces annales contiennent un chapitre contenant les partiels suivi d'un chapitre contenant les examens finaux ou de rattrapage. Toutes les solutions sont rassemblées dans un chapitre final.

Pour chaque examen, la liste des principaux utilitaires que l'on peut utiliser est récapitulée au début. Cette liste est d'ailleurs quelquefois fournie une semaine avant l'examen afin de donner le temps de lire les pages de manuel et ainsi voir tout ce qu'ils savent faire.

Chaque examen est composé de quelques exercices eux-mêmes structurés en questions souvent indépendantes. Chaque question indique quels fichiers livrer et quels types de tests seront effectués. Des hypothèses sont aussi formulées qui restreignent ces tests et rendent inutiles de se préoccuper de ce qui ne répond pas aux hypothèses. Par exemple, supposons que l'énoncé stipule que le script à livrer prend obligatoirement comme premier argument le nom d'un répertoire sur la ligne de commande ; une hypothèse indiquant que le script sera toujours invoqué avec un premier argument correct signifie que le script n'a pas à vérifier qu'il est bien appelé avec un répertoire existant, li-

sible, etc. Cela simplifie d'autant la programmation du script demandé.

Certains examens nécessitent de travailler sur des fichiers de données fournis en même temps que l'énoncé. Vous trouverez ces fichiers (compressés) sur le site des annales associé à cet enseignement, vous aurez (par convention) à les déployer dans le répertoire `/EnvDev/`.

1.3 Notations

Les interactions avec un ordinateur apparaissent comme suit :

```
% date | tee tmp/date
ven avr  7 09:32:17 MEST 2006
```

Les entrées apparaissent en gras pour les différencier des réponses de l'ordinateur. L'invite choisie pour l'interprète de commande est `%` : ce caractère est délivré par l'ordinateur pour vous inviter à entrer une commmande.

Les fichiers sont identifiés par un cartouche indiquant leur nom, une barre horizontale marque leur fin. Ainsi,

tmp/date

```
ven avr  7 09:32:17 MEST 2006
```

Les fichiers de type `Makefile` sont extrêmement sensibles à la présence de tabulations. Pour ces fichiers, les tabulations sont indiquées explicitement par des demi-rectangles inférieurs dont la longueur indique le saut nécessaire pour se déplacer au prochain taquet de tabulation. Ainsi,

tmp/Makefile

```
.PHONY: work clean

work : date

clean :
␣␣␣␣␣␣␣␣-rm date
date : Makefile
␣␣␣␣␣␣␣␣date > date

# fin de Makefile
```

Enfin, les longues lignes sont typographiées en plusieurs lignes, une petite main ☞ indique les cosmétiques passages à la ligne. Ainsi,

```
% if [[ -f tmp/date ]] ; then echo -n Le fichier est bien 'la ☞
☞ ' ; cat tmp/date ; else exit 1 ; fi
Le fichier est bien la ven avr  7 09:32:17 MEST 2006
```

1.4 Conseils

Pour travailler ces examens, munissez-vous d'un ordinateur avec les outils de développement Gnu usuels. Ils sont facilement installables sous Unix (Gnu/Linux ou Mac OS X) et sous Windows (avec Cygwin). Choisissez d'abord les partiels plus simples que les examens. La durée des partiels est de deux heures ; celles des examens est de trois heures. Ne lisez pas les solutions avant d'avoir programmé et testé les vôtres. En effet, l'algorithmique des programmes à réaliser ne présente pas de difficulté majeure, lire une solution simple ne peut que renforcer la reconnaissance de cette simplicité, et pourtant, le stress de l'examen rend, pour certains, cette simplicité inatteignable. Il n'y a malheureusement de compréhension (et de plaisir) que dans l'effort !

Entraînez-vous, entraînez-vous, entraînez-vous. Pensez que ce n'est pas après avoir dessiné correctement votre premier O que vous avez commencé à écrire des rédactions. Vous avez écrit des centaines de milliers de O au point que vous ne vous en souvenez plus. La programmation doit devenir aussi familière que l'écriture car c'est une écriture sauf qu'elle manipule des objets ayant des propriétés mathématiques plus assurées que les concepts flous de la vie courante.

Lisez attentivement les spécifications et notamment celles d'interface. Beaucoup d'erreurs sont commises par des programmes dont l'algorithmique est raisonnable mais dont la mise en œuvre n'est pas celle qui est demandée. Un programme peut, par exemple, prendre des données d'entrée soit par le flux d'entrée, soit par une variable d'environnement, soit par un argument de ligne de commande mentionnant le nom du fichier où se trouvent ces données. Des différences similaires existent en sortie. Lorsqu'un programme détecte qu'une situation est anormale, consultez la spécification pour savoir quoi faire. Ce peut être de s'arrêter immédiatement et de rendre un code de retour spécifié (ou simplement différent de zéro). Un message peut aussi être émis sauf mention contraire, sur le flux d'erreur afin de ne pas perturber les données (correctes) déjà émises. Utilisez le flux d'erreur pour la mise au point : ne polluez pas le flux de sortie standard !

Toutes les questions comportent une section intitulée « livraison » indiquant les fichiers attendus. Faites attention aux noms, qu'ils soient de fichiers, de variables d'environnement, de mots clés, etc. Un script qui doit se nommer `truc` n'est ni `truc.exe` ni `truc.sh`. Les fichiers à livrer sont rassemblés dans un répertoire nommé `envdev`.

En général, les programmes demandés doivent pouvoir tourner partout. Plus précisément, ils peuvent s'exécuter dans n'importe quel répertoire (même des répertoires où l'on ne peut écrire) et sur n'importe quelle machine dotée des mêmes utilitaires. On admettra toutefois que le répertoire `envdev` est toujours dans le `PATH` : il peut ainsi contenir vos propres utilitaires. En d'autres termes, cela veut dire, lorsque vous pre-

nez possession de l'ordinateur sur lequel vous allez travailler, que vous effectuez les opérations suivantes :

```
% mkdir envdev
% export PATH=$( pwd )/envdev:$PATH
```

Un des buts du cours est de vous faire prendre conscience que refaire une seconde fois certaines choses est une indication sûre qu'il y a matière à automatiser. Refaire une troisième fois les choses est une faute, dans la plupart des cas, il aurait fallu automatiser avant. Automatiser c'est justement le travail des informaticiens (par exemple à coup de **make**), pourquoi ne pas appliquer à eux-mêmes ce précepte ?

Vous devez tester les programmes que vous avez à écrire. S'ils ne sont pas justes du premier coup, il faut donc automatiser leur test. Mais si les programmes ne sont pas justes du premier coup, c'est qu'ils évoluent et que donc vous devez gérer ces versions (**rcs** ou **cvs** pourraient alors être vos amis) afin, notamment, de revenir promptement sur une ancienne version lorsque la nouvelle ne marche pas et que la fin de l'examen approche !

Vous attacherez un soin particulier au test de vos programmes et il est beaucoup, beaucoup, beaucoup, beaucoup, beaucoup plus important d'avoir quelques programmes qui fonctionnent qu'avoir touché à tout sans que rien ne fonctionne. Une stratégie utile est donc de procéder par petits pas. Écrivez d'abord un programme qui fonctionne sur le cas général puis ajoutez, selon un ordre intelligemment pensé, les différents cas particuliers mentionnés dans l'énoncé.

Si la notation aveugle ne se soucie absolument pas de la forme de vos programmes, en revanche, vous et vos enseignants auront à pâtir d'une mauvaise présentation. L'œil humain a de la peine à lire de longue lignes : limitez-vous à 80 colonnes. Les alignements à gauche (l'*indentation* en jargon) permettent de mieux appréhender la structure des programmes. Quand on sait que tous les éditeurs de texte décents indentent ou ré-indentent automatiquement, pourquoi s'évertuer à casser cet effet ?

Placez correctement vos fenêtres, ne les bougez pas, épargnez-vous les gestes superflus ! Quatre fenêtres (toutes visibles en même temps) devraient suffire : un visualiseur pour l'énoncé de la question courante, un grand éditeur de texte (avec éventuellement de multiples onglets ou sous-fenêtres), un interprète de commande **bash** pour tester la question courante et analyser les résultats, enfin un interprète de commande pour des travaux généraux et non répétitifs (consulter une page de manuel, lancer la visualisation d'une documentation en PDF ou HTML, tuer un processus qui boucle, modifier un droit d'accès à un fichier, etc.)

1.5 Conclusions

Nous espérons que ces annales sont un bon complément de cours et qu'elles vous permettront de vous entraîner efficacement.

Nous remercions tous nos collègues qui, depuis le début, ont lu, relu et amélioré ces épreuves.

1.6 Bibliographie

[QC02] Christian Queinnec et Emmanuel Chailloux. « Une expérience de notation en masse ». *TICE 2002 – Technologies de l'Information et de la Communication dans les Enseignements d'Ingénieurs et dans l'industrie – Conférences ateliers*, pp. 403–404, Lyon (France), novembre 2002. Institut National des Sciences Appliquées de Lyon. version complète disponible en `http://www-spi.lip6.fr/~queinnec/PDF/cfsreport.pdf`.

Chapitre 2

Partiels

A – Partiel – 1er décembre 2004

Les commandes un peu particulières qui peuvent être utilisées (mais ce n'est pas obligatoire) dans ce problème sont : **tar**, **gzip**, **date**, **md5sum**, **sed** ou **awk**. Un corrigé se trouve page 75.

▷ Exercice A.1 – Paquetages

Les fichiers de suffixe `.rpm` (pour *RedHat Package Manager*) sont des paquetages correspondant à des logiciels installables ou recompilables simplement. Le but de ce problème est de vous faire réaliser un mini-système de paquetage pour lesquels nous utiliserons le suffixe `.edp` (pour *EnvDev Package*).

Question A.1.1 Supposons que le fichier `liste` contienne une liste de noms de fichiers. Tous ces noms de fichiers ont des noms de chemin absolus. Les fichiers apparaissent un par ligne et sans blanc superflu. Chaque ligne s'achève par une fin de ligne. Voici le contenu d'un tel fichier nommé `liste` (dont on se servira par la suite) :

```
% cat liste
/EnvDev/usr/etc/dot.emacs
/EnvDev/usr/etc/tex/listAda.sty
/EnvDev/usr/doc/manLicence.ps
```

On dira qu'un fichier est de type `liste` lorsqu'il a un contenu de même structure que ce qui vient d'être décrit.

Écrire un programme, nommé `pack.sh`, prenant deux arguments sur sa ligne de commande, un nom de fichier à créer suivi d'un fichier de type `liste`. Cette commande crée l'archive compressée qui a pour nom le premier argument et qui contient les fichiers spécifiés par le second argument. L'archive sera créée par **tar** et compressée par **gzip**. Ainsi

```
% ls
liste
% ./pack.sh p.edp liste
```

Le paquetage qui vient d'être créé, l'archive compressée `p.edp`, existe dans le répertoire courant.

```
% ls
liste  p.edp
```

Le paquetage a pour contenu :

```
% tar tzf p.edp
EnvDev/usr/etc/dot.emacs
EnvDev/usr/etc/tex/listAda.sty
EnvDev/usr/doc/manLicence.ps
```

Notez que les chemins sont maintenant relatifs car **`tar`** ne mémorise pas les chemins absolus.

Livraison

– Le fichier exécutable `pack.sh` ainsi que les éventuels autres fichiers nécessaires à son bon fonctionnement.

Notation sur 3 points

– 3 points si votre programme `pack.sh` a le bon comportement.

Hypothèse: Le second argument est une liste correcte de fichiers qui existent tous. Cette liste contient au moins un fichier.

Hypothèse: Le contenu du flux d'erreur ne sera pas considéré.

Question A.1.2 L'archive compressée `p.edp` précédemment produite sera le paquetage qui nous servira d'exemple. On désire maintenant y ajouter des informations permettant de gérer ce paquetage. Écrire un programme, nommé `pack2.sh`, prenant les mêmes arguments que précédemment à savoir le nom du paquetage à créer et le fichier de type `liste`. Le paquetage créé contiendra, en plus, des méta-informations. Ces méta-informations seront stockées dans un répertoire que l'on nommera `EDP-INF` ; elles seront au nombre de trois :

– un fichier `EDP-INF/TOC` contenant la liste des noms des fichiers embarqués dans l'archive (ceux présents dans le fichier donné en second argument du programme `pack2.sh`). Ce fichier sera de type `liste`.

– un fichier `EDP-INF/DATE` contenant la date d'enregistrement. Cette date apparaîtra sous une forme numérique. Ainsi `200412010830` correspond au 1er décembre 2004 à 8h30 du matin.

– un fichier `EDP-INF/VERSION` contenant le chiffre 1 pour indiquer la version du système de paquetage.

Pensez à supprimer ce répertoire `EDP-INF` après (ou avant) usage (c'est plus propre après, c'est plus sûr avant). Ainsi,

```
% ./pack2.sh p.edp liste
% tar tzf p.edp
EDP-INF/DATE
```

```
EDP-INF/TOC
EDP-INF/VERSION
EnvDev/usr/etc/dot.emacs
EnvDev/usr/etc/tex/listAda.sty
EnvDev/usr/doc/manLicence.ps
```

Livraison

– Le fichier exécutable `pack2.sh` ainsi que les éventuels autres fichiers nécessaires à son bon fonctionnement.

Notation sur 3 points

– 3 points si votre programme `pack2.sh` a le bon comportement.

Hypothèse: Le second argument est une liste correcte, éventuellement vide, de fichiers qui existent tous.

Question A.1.3 Écrire un programme, nommé `install.sh`, prenant le nom d'un paquetage en argument ainsi qu'une option (`--prefix=`) indiquant le préfixe d'installation à utiliser. Cette option peut être placée avant ou après le nom du paquetage. Ainsi les deux commandes suivantes sont-elles équivalentes :

```
./install.sh --prefix=/tmp p.edp
./install.sh p.edp --prefix=/tmp
```

Si l'option est absente, elle sera équivalente à `--prefix=/`.

Le programme extrait les fichiers mentionnés dans `EDP-INF/TOC` et les copie dans le système de fichiers local à partir du répertoire indiqué par le préfixe d'installation. Ainsi pour installer les trois fichiers présents dans le paquetage `p.edp`, peut-on écrire :

```
% ls /tmp/EnvDev
ls: /tmp/EnvDev: No such file or directory
% install.sh --prefix=/tmp p.edp
% ls /tmp/EnvDev/usr/
doc    etc
```

Les fichiers sont installés à partir de `/tmp/` suivis de leur nom tel qu'il apparaissait dans la liste des fichiers soumis à la commande de création du paquetage. Ainsi le fichier `manLicence.ps` sera-t-il installé en :

```
/tmp/EnvDev/usr/doc/manLicence.ps
```

Les méta-informations, placées dans le répertoire `EDP-INF/` présent dans le paquetage, ne sont pas extraites. Pour donner un exemple un peu plus complet :

```
% ls /tmp/EnvDev
ls: /tmp/EnvDev: No such file or directory
% install.sh --prefix=/tmp p.edp
% ls -R /tmp/EnvDev/usr
/tmp/EnvDev/usr:
doc  etc

/tmp/EnvDev/usr/doc:
```

```
manLicence.ps

/tmp/EnvDev/usr/etc:
dot.emacs  tex

/tmp/EnvDev/usr/etc/tex:
listAda.sty
```

Indice : il peut être utile de consulter l'option `-C` de **`tar`** ou de faire des calculs à l'aide de **`pwd`** afin de mettre en relation le paquetage et le répertoire d'installation.

Livraison

– Le fichier exécutable `install.sh` ainsi que les éventuels autres fichiers nécessaires à son bon fonctionnement.

Notation sur 3 points

– 3 points si votre programme `install.sh` a le bon comportement.

Hypothèse: Votre script sera invoqué avec un ou deux arguments. Le paquetage sera un paquetage correct. Le répertoire indiqué par le préfixe d'installation (si mentionné) est réputé exister.

Question A.1.4 Écrire un programme, nommé `pack3.sh`, similaire à `pack2.sh` sauf que le fichier `EDP-INF/VERSION` contient maintenant le nombre 2 et que le fichier `EDP-INF/TOC` contient des lignes où le nom du fichier est précédé du résumé MD5 du fichier suivi d'un blanc.

Le résumé MD5 d'un fichier est un nombre de 128 bits tel que la plus légère des modifications (la modification d'un unique bit du fichier) change totalement le résumé.

Si des fichiers à empaqueter manquent, le programme `pack3.sh` retournera un code de retour erroné et mentionnera le problème sur son flux d'erreur (cf. exemple ci-dessous). La commande s'arrêtera à la première erreur détectée. Ainsi,

```
% md5sum /EnvDev/usr/etc/dot.emacs
5b4c8f44edbab9922f0134911230560a /EnvDev/usr/etc/dot.emacs
% ./pack3.sh p3.edp liste
% tar xzf p3.edp EDP-INF/TOC ; cat EDP-INF/TOC
5b4c8f44edbab9922f0134911230560a /EnvDev/usr/etc/dot.emacs
f689928b5f9b288e2b4326477cd04937 /EnvDev/usr/etc/tex/listAda.sty
147ac9d0193360cd87ac748ae9676d1b /EnvDev/usr/doc/manLicence.ps
% cat liste > liste3 ; echo /usr/foobar >> liste3
% ./pack3.sh p3.edp liste3
Fichier manquant: /usr/foobar
```

Livraison

– Le fichier exécutable `pack3.sh` ainsi que les éventuels autres fichiers nécessaires à son bon fonctionnement.

Notation sur 3 points

– 3 points si votre programme `pack3.sh` a le bon comportement.

Hypothèse: Le second argument est une liste correcte, éventuellement vide, de fichiers.

Question A.1.5 Écrire un programme, nommé `verify.sh`, prenant le nom d'un paquetage de version 2 en argument (ainsi qu'éventuellement une option `--prefix=`) et vérifiant s'il est installé c'est-à-dire si tous les fichiers qu'il décrit existent, sont au bon endroit et ont le même résumé MD5 (notez que l'on ne s'intéresse pas à la date de création du fichier mais seulement à son contenu). Si l'option `--prefix=` est absente, elle sera équivalente à `--prefix=/`.

La correction de la vérification est rendue dans le code de retour, les fichiers manquants ou modifiés seront signalés sur le flux d'erreur (l'ordre est indifférent) comme indiqué dans l'exemple qui suit :

```
% if verify.sh p.edp ; then echo OK ; else echo KO ; fi
OK
% touch /EnvDev/usr/etc/dot.emacs
% if verify.sh p.edp ; then echo OK ; else echo KO ; fi
OK
% echo coucou >> /tmp/EnvDev/usr/etc/dot.emacs
% rm /tmp/EnvDev/usr/etc/tex/*.sty
% if verify.sh --prefix=/tmp p.edp; then echo OK; else echo KO; fi
Fichier different: /tmp/EnvDev/usr/etc/dot.emacs
Fichier absent: /tmp/EnvDev/usr/etc/tex/listAda.sty
KO
```

Indice: On placera le fichier `EDP-INF/TOC` dans `/tmp/` et on l'effacera après usage.

Livraison

– Le fichier exécutable `verify.sh` ainsi que les éventuels autres fichiers nécessaires à son bon fonctionnement.

Notation sur 2 points

– 2 points si votre programme `verify.sh` a le bon comportement.
Hypothèse: L'argument est un paquetage correct.

Question A.1.6 Écrire un programme, nommé `uninstall.sh`, prenant le nom d'un paquetage de version 1 ou 2 ainsi qu'un éventuel préfixe (avec l'option `--prefix=`) et supprimant les fichiers installés concernés. Si l'option `--prefix=` est absente, elle sera équivalente à `--prefix=/`. Le code de retour indique si la suppression des fichiers concernés a bien eu lieu.

Puisqu'il s'agit de désintaller, la commande `uninstall.sh` tente d'effacer tout ce qu'elle peut. Lorsqu'un fichier (ou répertoire) est effacé et si le répertoire le contenant ne contenait que cet unique fichier (ou répertoire), ce répertoire sera également supprimé et ainsi de suite récursivement. Ainsi,

```
% ls /tmp/EnvDev/usr
doc   etc
% ./uninstall.sh --prefix=/tmp/ p.edp
% ls /tmp/EnvDev
ls: /tmp/EnvDev: No such file or directory
```

Livraison

– Le fichier exécutable `uninstall.sh` ainsi que les éventuels autres fichiers nécessaires à son bon fonctionnement.

Notation sur 3 points

– 3 points si votre programme `uninstall.sh` a le bon comportement.

Hypothèse: L'argument est un paquetage correct.

Question A.1.7 Réunir les quatre derniers programmes c'est-à-dire `pack3.sh`, `install.sh`, `uninstall.sh` et `verify.sh` en un unique programme, que l'on nommera `paquetage.sh`, qui peut incarner les quatre programmes précédents. Ce programme ne sera utilisé que sur des paquetages de version 2, il faut donc mettre à jour la commande `install.sh`. La commande `paquetage.sh` prend en premier argument le nom de la commande à incarner à savoir `pack`, `install`, `verify` ou `uninstall`. Les arguments qui suivent sont ceux de la commande incarnée.

Ainsi les quatre commandes qui suivent correspondent-elles aux anciennes commandes (l'option `--prefix` n'est pas mentionnée pour ne pas alourdir les exemples mais elle ne peut apparaître qu'en dernière ou avant-dernière position) :

```
% paquetage.sh pack p.edp liste  ≡ pack3.sh p.edp liste
% paquetage.sh install p.edp     ≡ install.sh p.edp
% paquetage.sh verify p.edp      ≡ verify.sh p.edp
% paquetage.sh uninstall p.edp   ≡ uninstall.sh p.edp
```

Livraison

– Le fichier exécutable `paquetage.sh` ainsi que les éventuels autres fichiers nécessaires à son bon fonctionnement.

Notation sur 3 points

– 3 points si votre programme `paquetage.sh` a le bon comportement.

Hypothèse: Le premier argument d'appel à `paquetage.sh` est supposé correct c'est-à-dire l'un des quatre identificateurs permis.

B – Partiel – 1er avril 2005

Parmi les utilitaires Unix qui peuvent être employés à bon escient dans cet examen, citons **`bash`**, **`sed`**, **`cut`**, **`awk`**, **`sort`**, **`tr`**. Un corrigé se trouve page 85.

Afin de fixer les idées, voici un schéma représentant l'organisation du répertoire `envdev` :