CFSkit
ou , comment concevoir et mettre en œuvre une notation en masse

Id: howtocfs.bk,v 1.37 2004/04/15 15:40:35 queinnec Exp

Christian Queinnec
Université Paris 6 — Pierre et Marie Curie
LIP6, 4 place Jussieu, 75252 Paris Cedex
France – Mél: `Christian.Queinnec@lip6.fr`

Ce document décrit CFSkit, un ensemble de programmes destinés à la notation en masse. Les premières expériences ont été publiées dans l'article [QC02]. Les programmes et cette documentation peuvent être récupérés sur le site de l'auteur en http://youpou.lip6.fr/queinnec/Programs/cfskit-latest.tgz

Le CFSkit évolue au gré des notations en masse qu'il réalise : la bibliothèque standard s'étoffe et se dépouille pour être plus générale et plus réutilisable, les programmes résistent mieux aux actes de malveillance.

Le CFSkit est assez paramétrable, tous les déploiements possibles n'ont pas encore été imaginés. Cette documentation devrait évoluer au fil de vos remarques.

Table des matières

1 Présentation générale

La « notation en masse » part d'un répertoire contenant un ensemble de « copies » d'étudiants à corriger. Ces copies sont techniquement des fichiers d'archive compressée (des *tar.gz* en jargon) contenant les fichiers (données, textes de programmes, binaires ou quoi que ce soit qui ait été demandé aux étudiants). La notation en masse consiste à produire, pour chacune de ces copies, une « copie annotée » c'est-à-dire notée et accompagnée d'une verbalisation (en HTML) justifiant cette note.

Les copies annotées sont destinées à être rendues consultables, via le ouèbe, par les étudiants, ce qui permet de simplifier le processus de consultation des copies. Les notes partielles ou totales peuvent être extraites des copies annotées, mises en base et affichables sur le ouèbe. D'autres faits peuvent également être collectés comme les langages utilisés, les temps de calcul, etc.

Les programmes composant le CFSkit ont été plusieurs fois utilisés et les tests récurrents (écrits en `sh`) ont été factorisés en une bibliothèque simplifiant l'écriture de nouveaux tests. Elle est décrite en section 2 mais est destinée à être enrichie par vos contributions.

Les programmes du CFSkit ont été pensés pour permettre des corrections en parallèle sans interférence et limitant les risques liés aux programmes malveillants que pourraient écrire les étudiants. Une machine de confinement, bien gérée, isolée et sans serveur est à votre disposition à l'UFR pour noter en masse.

Des conseils pour écrire des tests apparaissent en section 3. Tourner CFSkit pour noter ou pour mettre au point ses tests requiert de comprendre comment déployer ses programmes. Cet art est décrit en section 4.

Les programmes du CFSkit sont gouvernés par des dizaines de paramètres. Cette documentation explique les principaux d'entre eux, les commentaires dans les sources (en `sh`, C, Makefile ou Perl) expliquent les autres.

Un examen à notation automatique procure quelques avantages :
– équité de la notation,
– possibilité de traiter de vastes nombres d'étudiants à investissement constant,
– annales dynamiques (les étudiants peuvent refaire les examens et obtenir leur notation).

1.1 Algorithme général

La notation en masse est lancée par la commande `run.sh`, la progression de la notation est indiquée sur le flux d'erreur. La syntaxe d'appel est la suivante :

```
run.sh [-n] [-p] fichiers|répertoires ...
```

Lorsque les copies sont ramassées par Delivery_Builder, l'option `-n` permet de ne noter, par étudiant, que la plus récente de ses soumissions. Les arguments suivants sont soit des fichiers archives de suffixe `.tar.gz`, soit des répertoires contenant ces mêmes fichiers archives. À la place de l'option `-n`, la variable `RT_NEWEST` peut être mise à `true`.

L'algorithme général de notation est le suivant :

```
1 Pour tout étudiant
2   Pour toute copie de cet étudiant (ou seulement la dernière)
3    Pour chaque question de l'énoncé
4      Pour chaque façon d'éprouver la réponse fournie
5        noter partiellement la copie de l'étudiant
6        en verbalisant cette notation
7      consolider les notes partielles en une note pour la question
8    collecter ces annotations en une copie corrigée
```

En ligne 1, les copies des étudiants sont trouvées par CFSkit à partir des arguments d'appel à la commande `run.sh` soit par indication d'un (ou plusieurs) répertoire(s) soit par énumération explicite des noms des copies.

En ligne 2, la copie de l'étudiant est installée dans un répertoire de travail dont le nom (techniquement un UUID) est garanti unique ce qui assure l'indépendance de notations réalisées en parallèle. Ce répertoire devient le HOME de l'étudiant. Un second répertoire est créé qui contiendra la copie corrigée à savoir le flux de sortie standard du programme interne `runalltests.sh`. Cette copie annotée est l'unique résultat et de nombreuses précautions sont prises pour éviter que les programmes de l'étudiant puissent perturber ce résultat.

Le script `runalltests.sh` note une unique copie.

En ligne 3. Un énoncé est une suite de questions. À un énoncé est associé un répertoire (fourni à `run.sh` par une variable d'environnement : `RT_TESTS_DIR`). À chaque question est associé un répertoire contenant les différents tests à effectuer sur les fichiers fournis par l'étudiant. Les questions sont traitées successivement en ordre alphabétique. Pour rappeler que ce sont des questions, leur nom doit commencer par `Q` ou contenir la chaîne `-Q` dans leur nom.

Voici la structure d'un énoncé */home/tartempion/CFS* à 2 questions[1] :

```
/home/tartempion/CFS/Q1/
                    /Q2/
```

En ligne 4, on note la réponse d'un étudiant à une question. C'est le script `runtests.sh` qui s'occupe de cela. Le répertoire associé à la question est scruté et tous les exécutables (les « tests ») dont le nom commence par deux chiffres (et qui ne se terminent pas par un tilde) sont exécutés dans l'ordre. Par exemple, si le répertoire */home/tartempion/CFS/Q2/* contient :

```
/home/tartempion/CFS/Q2/00construire
                       /10test
                       /11langages
                       /input1
                       /output1
                       /Makefile
                       /PROLOGUE
                       /EPILOGUE
                       /TITLE
```

Une ligne est engendrée dans la copie annotée décrivant le test en cours. Le nom du répertoire nomme le test sauf s'il existe un fichier de nom TITLE dans ce répertoire. Avant tout test, le texte (HTML) du fichier PROLOGUE si présent dans le répertoire est émis dans la copie annotée.

En ligne 5 : les tests (usuellement des scripts exécutables écrits en `sh`) `00construire`, puis `10test` suivi de `11langages` seront tous successivement exécutés. Un script s'arrête à la première erreur mais les tests suivants sont néanmoins exécutés. Un test peut être écrit en `sh`, en Perl, en ce que l'on veut. Les tests ne prennent aucun argument sur la ligne de commande, ils ne sont paramétrés que par les variables d'environnement.

[1] S'il y a plus de 9 questions, il est préférable de prendre des noms tels que Q01, Q02, etc.

Les tests sont limités en temps, en nombre de processus, en octets écrits en fichiers, etc. Des jeux de limites de plus en plus restreints sont appliqués à la correction d'une copie, d'une question et aux programmes des étudiants.

Les fichiers qui ne sont pas des tests sont souvent des données utilisées par les tests. Si un fichier *Makefile* existe alors, la commande `make init` sera invoquée avant la notation (et donc avant la ligne 1). Le rôle usuel de ce *Makefile* est de, par exemple, construire le fichier *output1* à partir de *input1*.

En ligne 6 : la sortie standard d'un test est ajoutée à la copie annotée. La sortie d'erreur est accumulée dans un fichier qui est ensuite ajouté à la copie annotée. Ces deux sorties sont encadrées de balises leur conférant des apparences différentes (en noir pour la sortie normale, en rouge pour la sortie d'erreur).

Un test est nommé par son nom de fichier ou par les mots figurant sur une ligne débutant par `#TITLE` dans le fichier même.

En ligne 7 : Une conclusion est ajoutée lorsque le dernier test d'une question s'achève. Cette conclusion présente la somme des points glanés (par chaque test) dans la question. Après cette conclusion est émis le contenu du fichier EPILOGUE du répertoire si présent.

En ligne 8 : Une conclusion est ajoutée lorsque la dernière question de l'énoncé est notée. Cette conclusion présente la somme des points glanés (par chaque question) en tout. La phrase « Cette somme ne tient pas compte des éventuels points de bonification qui ne pourraient être calculés que sur l'ensemble de tous les programmes. » est systématiquement ajouté après le total de points en fonction de la valeur de la variable `RT_DISPLAY_DISCLAIMER` (par défaut `true`.

2 Bibliothèque de test

La bibliothèque de tests (le fichier *runtestlib.sh*) contient de nombreuses fonctions `sh` et rend plus aisée l'écriture de nouveaux tests. Ayant été utilisée, à ce jour (fin 2002), par 4 CFS et seulement 2 DTH, elle est biaisée en faveur des CFS.

La copie annotée est un fragment XML propre à être visualisé par tout navigateur. Les balises utilisent des attributs `class` qui peuvent trouver leur interprétation dans une feuille de style ou *.css*.

2.1 Environnement

Chaque test est un exécutable dont le flux de sortie suffixe la copie annotée. Les tests sont exécutés dans le répertoire HOME de l'étudiant où se trouvent les fichiers composant sa copie. Les tests ne prennent aucun argument sur leur ligne de commande et n'ont pas de flux d'entrée. Ils sont un peu confinés pour limiter les erreurs d'enseignants.

Tout test peut compter sur la présence d'un certain nombre de variables d'environnement lui permettant de s'adapter au contexte, de déterminer où stocker des données pour de futurs tests (jamais dans le répertoire courant).

Voici ces variables :

- `RT_MARK_DIR` est le répertoire contenant les sources du CFSkit.
- `RT_UUID` est le nom unique de la copie en cours de correction. Ce nom n'est attribué qu'une unique fois et n'est jamais réutilisé. Il ressemble à quelque-chose comme `e49bc795-d902-42dd-a327-d3f7ea389ebe`.
- `RT_TGZ` est le nom de la copie en cours de notation. Si ce nom est attribué par Delivery_Builder, il ressemble à quelque chose comme *p6lip544_p6lip234_001.tar.gz* ou *p6lip544_NOBODY_001.tar.gz* dans le cas d'un monôme.
- `RT_Q` est le nom de la question en cours de correction.
- `RT_THE_TEST_DIR` est le répertoire contenant les tests de la question courante (et les fichiers d'accompagnement).
- `RT_TMP` est un répertoire où stocker des données temporaires sans interférence. Il est unique pendant toute la correction de la copie. Ce répertoire est normalement détruit lorsque la correction de la copie s'achève.

Toutes ces variables ont des noms débutant par le préfixe `RT_` afin d'être inaccessible aux programmes des étudiants.

2.2 Génération d'XML

Les fonctions suivantes ont pour rôle premier d'engendrer des fragments d'XML pour exprimer des faits de différentes natures. Toutes ces commandes ont des noms débutant par le préfixe `rt_` afin d'être inaccessible aux programmes des étudiants.

```
rt_echo mot mot ...
```

Cette commande permet d'émettre des suites de mots. En général, cette commande verbalise le test qui va avoir lieu ou commente les résultats obtenus. Le résultat apparaît sans décoration superflue (noir sur fond blanc classe ECHO).

```
rt_warning mot mot ...
```

Cette commande permet d'émettre des suites de mots en guise d'avertissement. Personnellement j'utilise un avertissement pour signaler un fait que je réprouve mais que, néanmoins, je tolère. Les avertissements apparaissent en orange (classe WARNING) encadrés de crochets carrés.

```
rt_error mot mot ...
```

Cette commande permet d'émettre des suites de mots pour signaler une erreur. Les erreurs sont notifiées en rouge (classe ERROR)

```
rt_file_echo fichier
```

Cette commande émet le contenu du fichier. Il apparaitra en bleu sombre sur fond jaune (classe FILE). Les caractères spéciaux d'HTML (<, > et &) sont correctement transcodés.

```
rt_section mot mot ...
rt_subsection mot mot ...
rt_subsubsection mot mot ...
```

Ces commandes permettent de structurer le flot de sortie en sections (balise h2) sous-sections (balise h3) et sous-sous-sections (balise h4).

```
rt_start_stdout
rt_stop_stdout
rt_start_stderr
rt_stop_stderr
```

Lorsqu'on souhaite montrer ce que produit un programme sur sa sortie standard ou sa sortie d'erreur, on utilise ces commandes. La sortie d'erreur apparait en rouge sur fond jaune (classe STDERR), la sortie standard apparaît en (classe STDOUT). Ces commandes encadrent les commandes dont la sortie (resp. la sortie d'erreur) est à orner. Normalement, la sortie d'erreur d'un script, si non vide, apparaît automatiquement correctement ornée en fin de verbalisation du script.

2.3 Génération de faits

Les fonctions suivantes permettent d'engendrer des faits, des notes, des informations destinées aux programmes qui exploiteront la copie annotée.

```
rt_win perl-expression
```

Cette commande indique un nombre de points (positifs s'ils sont gagnés, négatifs s'ils représentent une pénalité). Les points apparaissent en blanc sur fond bleu (classe MARK). L'argument est une expression arithmétique en Perl, la syntaxe est usuelle. La note est tronquée à deux décimales après la virgule. Voici quelques exemples :

```
rt_win 2                    # gagne 2.00 points
rt_win 2.3-1                # gagne 1.30 points
rt_win 2.3456 - 1           # gagne 1.34 points
rt_win '(-2.4 + (3*4)/2)'   # gagne 3.60 points


rt_textual_fact label fact
rt_numeric_fact label fact
```

Ces commandes permettent d'émettre un fait textuel ou numérique. Elles engendrent une balise XML nommée FACT et contenant des attributs mentionnant l'UUID de la copie, la question courante, le label et le fait (textuel ou numérique). Ces commandes sont plutôt destinées à collecter de l'information sur la copie (langage employé, durée de test, etc.) qui ne participent pas à la note mais qui peuvent être utile à des fins statistiques.

2.4 Vérifications courantes

Les fonctions suivantes verbalisent des tests courants.

`rt_exist fichier`

Cette commande vérifie (et verbalise) que le fichier existe et est non vide. La commande exécute un `exit` avec un code d'erreur (différent de zéro donc) si le fichier n'existe pas ou qu'il est vide. Non rattrapé, le script courant sera alors interrompu.

`rt_executable_exist fichier`

Cette commande vérifie (et verbalise) que le fichier existe, est non vide et exécutable. La commande exécute un `exit` avec un code d'erreur (différent de zéro donc) si le fichier n'existe pas ou qu'il est vide ou qu'il n'est pas exécutable. Non rattrapé, le script courant sera alors interrompu.

`rt_grep [options] regexp fichier`

Cette commande vérifie (et verbalise) que le fichier existe et qu'il contient une ligne satisfaisant l'expression rationnelle *regexp*. Si ce n'est pas le cas, le script courant sera interrompu. Les options peuvent être spécifiées par un unique mot débutant par un tiret.

`rt_build programme script`

Pour les CFS, cette commande tente de reconstruire un exécutable nommé *programme* à partir d'un script nommé *script*. Si l'exécutable à produire existait déjà avant, il est renommé. Usuellement, le programme est nommé *nom*`.exe` et le script `compil-`*nom*`.sh` c'est ce qu'assumait l'ancienne commande `rt_rebuild` maintenant obsolète.

`rt_recuperate executable`

Pour les CFS, cette commande s'utilise souvent après la précédente. Elle permet de continuer les tests avec l'*exécutable* s'il a été renommé par `rt_rebuild` et qu'aucun exécutable n'a été reconstruit par `rt_rebuild`. Un avertissement est produit mais cette pratique est contestable car elle peut encourager le plagiat d'exécutables.

`rt_guess_file cible fichier ...`

Cette commande (due à Christophe Gonzales) permet de déterminer le fichier dont le nom est le plus proche (au sens de la distance de Levenshtein) de « *cible* » parmi les fichiers figurant comme arguments de la commande. Le nom de fichier, si trouvé, est produit sur le flux de sortie. La commande retourne zéro si un nom a été trouvé, non-zéro autrement. Le paramètre `RT_TYPO_THRESHOLD` (par défaut 3) fixe le nombre maximal d'erreurs admises.

`rt_guess_language`

Cette commande s'utilise après la commande `rt_build` et tente de déterminer le(s) langage(s) utilisé(s) pour la question courante. Cette connaissance si acquise est incorporée comme un fait à l'aide de la commande `rt_textual_fact`.

`rt_confiner [--cpu=n] [--output=t] [--ignore-return-code] programme argument ...`

Cette commande permet d'exécuter le programme (un fichier exécutable) et ses arguments dans un environnement confiné en temps, en nombre de processus, en écriture en fichiers sur disque etc. Plusieurs paramètres sont possibles qui permettent de régler plus finement le nombre de secondes CPU à donner (il ne peut excéder le confinement courant) et le nombre de Koctets écrivables en fichiers sur disque (l'unité est le Koctet). Les limites actuelles sont spécifiées dans le fichier *limits.sh*.

Par défaut, le code de retour est vérifié et fait l'objet d'un avertissement si différent de zéro. Cette vérification n'est pas effectuée si l'option `-ignore-return-code` est présente. Cette vérification peut être effectuée plus tard avec la commande `rt_code ' cat ${RT_TMP}/.lastcode '` à condition que ce fichier (créé par la commande `rt_confine`) n'a pas disparu.

Tout programme de l'étudiant doit impérativement n'être exécuté qu'au travers de cette commande. Toutes les variables, fonctions et autres informations disponibles pour l'enseignant (et dont les noms commencent par `rt_` ou `RT_`) sont retirées de l'environnement passé au programme de l'étudiant.

`rt_tee taille fichier`

Ce filtre est similaire au filtre `tee` : il copie son flux d'entrée sur son flux de sortie tout en faisant une copie dans le *fichier*. Le premier paramètre limite la copie à un nombre d'octets spécifié. Ce nombre d'octets peut utiliser les unités `k` ou `m` (casse non signifiante). Ce filtre n'est intéressant que parce qu'UNIX ne confine que le nombre d'octets écrits sur disque et non pas le nombre d'octets émis sur le flux de sortie standard.

Tout programme de l'étudiant doit avoir sa sortie limitée que ce soit pour les programmes qui bouclent ou pour des programmes malveillants. Le flux d'erreur étant détourné en fichier est de fait confiné.

2.5 Comparaisons

Quelques fonctions de comparaison existent.

`rt_diff [options] fichier1 fichier2`

Cette commande compare (et verbalise la comparaison) des deux fichiers mentionnés en arguments. Elle utilise `diff`. Les différences sont affichées en rouge (classe STDERR) et le script courant est interrompu lorsque les fichiers diffèrent. Les options peuvent être spécifiées par un unique mot débutant par un tiret. Par défaut les options sont `-abBcwi`.

`rt_prefix_diff [options] fichier1 fichier2 n`

Cette commande compare (et verbalise la comparaison) les *n* premières lignes des deux fichiers mentionnés en arguments. Elle utilise `diff`. Les différences sont affichées en rouge (classe STDERR) et le script courant est interrompu lorsque les fichiers diffèrent. Les options peuvent être spécifiées par un unique mot débutant par un tiret. Par défaut les options sont `-abBcwi`.

`rt_compare points fichier-enseignant fichier-etudiant`

Cette commande compare (et verbalise la comparaison) les deux fichiers mentionnés en arguments. Elle utilise la distance de Levenshtein qui calcule le nombre de lignes à supprimer, remplacer ou insérer pour convertir le premier fichier en le second. Il vaut mieux que le fichier *fichier-enseignant* ne soit pas vide ! Le calcul étant quadratique, il est limité à 15 secondes, le résultat *d* peut donc ne pas être obtenu auquel cas il faut probablement se rabattre sur une autre comparaison : `rt_compare_with_diff`.

Si la comparaison a donné un résultat alors l'étudiant gagne une fraction de *points*. La formule exacte est, en Perl (où `szt` est le nombre de lignes du *fichier-enseignant*, `szs` le nombre de lignes du *fichier-étudiant* et *d* la distance de Levenshtein calculée) :

```
my $sz = (($szs>$szt)?(2*$szt):($szs+$szt));
my $r = $points * ( 1 - $d/$sz);
print (($r>0)?$r:0);
```

Comme indiqué, la note finale n'est jamais négative. La commande `rt_compare` peut prendre l'option (en premier argument exclusivement) `-ignore-bad-exit-code` qui ne tient pas compte des avertissements produits par CFSkit lorsque le programme inspecté se termine par un code de retour différent de zéro.

```
rt_compare_with_diff points penalité fichier-enseignant fichier-etudiant
```

Cette commande compare (et verbalise la comparaison) les deux fichiers mentionnés en arguments. Elle utilise `diff -abBwi` pour être rapide et calcule, en gros, le nombre de lignes *N* qui diffèrent. La note est alors calculée suivant la formule, en Perl :

```
my $d = $total - $penalty * $N;
print (($d>0)?$d:0);
```

Comme indiqué, la note finale n'est jamais négative.

2.6 Recommandations

Les tests sont tournés dans le répertoire contenant les fichiers de l'étudiant. Les utilitaires du CFSkit sont à invoquer avec le chemin `RT_MARK_DIR`. Il est d'usage de charger la bibliothèque de base au début d'un test :

```
source $RT_MARK_DIR/runtestlib.sh
```

Lorsqu'un fichier temporaire doit être produit, il faut toujours le créer dans le répertoire `RT_TMP` afin de ne pas perturber ni le répertoire de l'étudiant ni les corrections qui se déroulent en parallèle.

Toujours confiner l'exécution des programmes de l'étudiant ! Toujours limiter la sortie standard des programmes de l'étudiant ! Ces deux admonestations énoncées pour votre propre bien conduisent donc à écrire, lorsque l'on veut tester un *programme* d'étudiant, des phrases comme :

```
rt_confiner ./programme 2>${RT_TMP}/err | rt_tee 10k ${RT_TMP}/out
```

Voici un exemple d'emploi de quelques unes des primitives précédentes :

```
01 #! /bin/bash
02 source $RT_MARK_DIR/runtestlib.sh
04 rt_echo Je vais tester votre programme avec les entrées suivantes:
05 rt_file_echo ${RT_THE_TEST_DIR}/input1
06 rt_echo "Votre programme donne:"
07 rt_start_stdout
08 rt_confiner ./programme 2>${RT_TMP}/err | rt_tee 10k ${RT_TMP}/out
09 rt_stop_stdout
10 rt_grep 'le resultat est [0-9]' ${RT_TMP}/out
```

```
11 rt_win 0.5
12 if [ -s ${RT_TMP}/err ]
13 then rt_error "Oh la la, il y a des choses etranges:"
14      rt_start_stderr
15      cat ${RT_TMP}/err && rm ${RT_TMP}/err
16      rt_stop_stderr
17 else rt_win 0.5
18 fi
```

Le script est exécutable par `bash` en ligne 1. La bibliothèque standard est lue en ligne 2. On verbalise le test que l'on va effectuer et le contenu du fichier d'entrée en lignes 4 et 5. On préfixe la sortie standard du programme de l'étudiant en ligne 6 et 7, le programme est tourné, confiné, en ligne 8, sa sortie apparaîtra dans la copie annotée mais est également stockée dans le fichier *out* (dans le répertoire temporaire approprié) qui est limité à 10Koctets. On ferme la balise indiquant que la sortie du programme de l'étudiant est terminée en ligne 9. On vérifie, en ligne 10, que la sortie contient bien une chaîne indiquant que le « résultat est » un certain chiffre. Si c'est le cas (ligne 11), on gagne 1/2 point. Si ce n'est pas le cas, le script est arrêté et on ne gagne bien sûr rien. Si la sortie d'erreur n'est pas vide (ligne 12) on émet un message d'erreur (ligne 13) puis l'on affiche le contenu du fichier contenant la sortie d'erreur (lignes 14–16) autrement on gagne encore un demi-point (ligne 17).

2.6.1 Remarques

De nombreuses fonctions `rt_` vérifient, comparent et invoquent `exit` en cas d'anomalie plutôt que de retourner un code d'erreur. La raison est que les tests sont plutôt vus comme étant courts et indépendants : dès que le comportement observé n'est pas celui attendu, le test est abandonné. Si l'on veut néanmoins utiliser une telle commande sans sortir du test courant, il faut rattrapper la sortie et donc exécuter la fonction dans un sous-processus. Par exemple, pour effectuer tous les tests suivants et ne pas s'arrêter au premier qui ne fonctionne pas, on écrira :

```
for f in *.txt
do
    ( OUT=‘basename $f .txt‘.out
      ${PROGRAM}.exe < $f > $RT_TEMP/$OUT
      rt_diff $OUT $RT_TEMP/$OUT )
done
```

De plus en plus, j'adopte l'organisation suivante qui permet d'améliorer la réutilisabilité des tests (l'idéal étant (peut-être) de construire des corrections automatiques en glissant-déposant les tests que l'on souhaite dans le bon répertoire). Dans le répertoire *tests/* se trouvent les fichiers suivants :

- *common.sh* qui contient des fonctions `bash` qui seront communes à tous les tests. C'est une bibliothèque spécialisée pour l'examen à noter.
- *globals.sh* qui ne contient que des variables exportées permettant d'adapter les bibliothèques à la machine de notation : quel compilateur Java utiliser, etc. Ce fichier est automatiquement inclus par le précédent.

Dans chaque répertoire correspondant à une question, se trouve un fichier *locals.sh* contenant les variables exportées particulières à la question : le nom du script de regénération, le nom de l'exécutable à produire, etc.

Tous les tests débutent donc par :

```
#! /bin/sh
# Charger la bibliotheque de fonctions communes:
source ${RT_MARK_DIR}/runtestlib.sh
# Charger les variables communes:
source ${RT_TESTS_DIR}/common.sh
# Charger les variables communes a la question:
```

```
source ${RT_THE_TEST_DIR}/locals.sh

...
```

3 Rédaction d'une épreuve

Pour avoir rédigé plusieurs examens à notation automatique, voici quelques conseils issus de mes expériences. Le point le plus important est que toute erreur (d'énoncé ou de correction) se paie au centuple (si l'on a plus de cent copies bien sûr !). Il est donc important d'être minutieux avant l'examen.

3.1 Questions

Voici les étapes menant à l'écriture d'une question.

1. Rédigez l'énoncé de la question. Spécifiez finement ce qui doit être fait, ce qui doit être remis, comment cela sera testé et noté. Regardez par exemples les énoncés des CFS[2].
2. Rédigez une ou plusieurs solutions (des justes, des fausses, des médiocres) (les vôtres, celles de vos amis) : cela permet d'affiner la spécification. Empaquetez ces solutions comme si c'étaient des copies d'étudiants (je les nomme des « pseudo-copies » et il y a des entrées prédéfinies dans les Makefile du CFSkit qui font cela automatiquement).
3. Rédigez les tests associés à la question (cela permet d'affiner encore les spécifications et d'adjoindre de nouvelles solutions). Passez les pseudo-copies d'étudiants et corrigez les tests, les solutions et les énoncés.

Il faut entre 2 et 4 heures par question environ.

3.2 Organisation

Les étapes précédentes peuvent être simplifiées en organisant ses répertoires. Je procède ainsi :

1. Créer un répertoire correspondant au contrôle à corriger automatiquement et les sous-répertoires pour ses différents sous-composants :

   ```
   mkdir MonExam/
   mkdir MonExam/src/
   mkdir MonExam/exam/
   mkdir MonExam/site/
   mkdir MonExam/pseudos/
   mkdir MonExam/tests/
   mkdir MonExam/tests/Q1
   mkdir MonExam/tests/Q2
   ```

   Le répertoire *src* contient les programmes du CFSkit. Ce peut être un simple lien vers ceux-ci. Le répertoire *exam* contient les sources de l'énoncé (en LaTeX ou autre). Le répertoire *site* contient les pages HTML ou PHP utiles pour parcourir les résultats de notation. Le répertoire *pseudos* contiendra les pseudo-copies, le répertoire *tests* contiendra les répertoires correspondant aux diverses questions (ici *Q1* et *Q2*).

   Tant que vous y êtes, écrivez un premier *Makefile* dans *MonExam/* pour automatiser certaines des opérations qui vont suivre.
2. Écrivez un bout d'énoncé (cf. annexe A) concernant la première question dans *exam* et placez-y un *Makefile* pour regénérer cet énoncé.

[2] `http://www.infop6.jussieu.fr/licence/2002/cct/`

3. Écrivez une première solution pour la première question. Pour cela, créez un sous-répertoire de *pseudos* où vous stockerez les fichiers demandés à l'étudiant. J'écris en général une réponse vide, une ou plusieurs réponses parfaites, et une ou plusieurs réponses intermédiaires correspondant à ce que j'intuite être des réponses qui seront communes et donc à considérer. J'indique, en général, dans le nom de ces pseudo-copies la question qu'elles traitent.

   Ainsi, crée-je et remplissé-je les répertoires (les noms des pseudo-copies débutent par le préfixe `p6lip` afin de ressembler aux copies qu'archivera Delivery_Builder. Attention aussi au blanc souligné qu'introduit Delivery_Builder !) :

   ```
   mkdir MonExam/pseudos/p6lip1_Q1_nul
   mkdir MonExam/pseudos/p6lip2_Q1_parfait
   mkdir MonExam/pseudos/p6lip3_Q1_moyen
   ```

   Dans chacun de ces répertoires, j'écris les fichiers stipulés par la question ainsi que le *Makefile* permettant d'éprouver ces programmes comme pourraient le faire les étudiants les mieux formés.

4. Écrivez un premier test dans *tests/Q1*, par exemple :

   ```
   emacs MonExam/tests/Q1/10debut
   ```

   Dans ce test, vous compilez les fichiers, les testez, etc. S'il y a des calculs à effectuer avant notation, indiquez-les dans un *Makefile* local avec une règle nommée `init`.

5. Il est temps de commencer à tester la notation automatique. Pour cela vous pouvez utiliser les fichiers prédéfinis dans CFSkit et ajouter dans *MonExam/Makefile* (Nota : les noms qui suivent sont ceux d'un exemple récent) :

   ```
   RT_SESSION      =       mlo-2002Nov
   CFSKIT_DIR      =       ${HOME}/Cours/MLO/Exam/2002Nov/src
   RT_LOG_DIR      =       /tmp/${RT_SESSION}/logs
   RT_RUN_DIR      =       /tmp/${RT_SESSION}/run
   LOCAL_DB        =       tmpmlo
   LOCAL_DBPREFIX  =       mlo2002nov
   include src/run.mkf
   ```

   Par défaut, cette entrée cherche à archiver les notes et faits produits dans une base de données Postgresql (pour les problèmes de base de données cf. infra). Elle transforme chaque sous-répertoire de *pseudos* en une copie (une archive compressée comme si effectuée par Delivery_Builder). Si une pseudo-copie nécessite de tourner quelque code avant d'être empaquetée, il doit y avoir une règle nommée `init` dans son *Makefile* local.

   Voici la signification des variables du *Makefile* : la variable `RT_SESSION` nomme l'épreuve, on indique où se trouve le répertoire du CFSkit dans la variable `CFSKIT_DIR`, où seront archivés les fichiers produits par la notation automatique (dans */tmp/mlo-2002Nov/logs/*), où seront tournées les pseudo-copies (dans */tmp/mlo-2002Nov/run/*), quel sera le nom de la base de données Postgresql qui sera utilisée pour recueillir les notes (j'utilise ici une base nommée `tmpmlo` différente de toutes mes autres bases) et quel sera le préfixe des tables qui seront engendrées (afin de ne pas polluer les tables qui pourraient déjà exister dans la base). Enfin on inclut le fragment de *Makefile* tout prêt dans le *Makefile*. Pour tous les répertoires, n'utilisez que des noms absolus !

   On note les pseudo-copies avec la commande :

   ```
   make pseudos.mark
   ```

6. Vous pouvez également (et c'est la méthode que je trouve la plus agréable) lancer un petit serveur HTTP pour vous aider à re-noter une copie (ou toutes les copies) ou inspecter les résidus d'exécution (les copies annotées ou le répertoire temporaire ou le répertoire de l'étudiant). Pour cela, lancer la commande (qui utilise également la base de données Postgresql (pour les problèmes de base de données cf. infra)) :

   ```
   make x.run.re-mark.server &
   ```

   Une fenêtre sera créée qui hébergera les sorties du serveur. Les premières lignes vous indiqueront quelle est l'URL à utiliser dans votre navigateur. Pour l'exemple courant, c'est :

   ```
   http://127.0.0.1:56001/tmp/mlo-2002Nov/logs/index.html
   ```

   Vous obtiendrez une fenêtre comme indiquée en figure 1 (il n'y a là qu'une unique copie et une unique question). La table est automatiquement recalculée et s'adapte au nombre de copies et au nombre de questions.

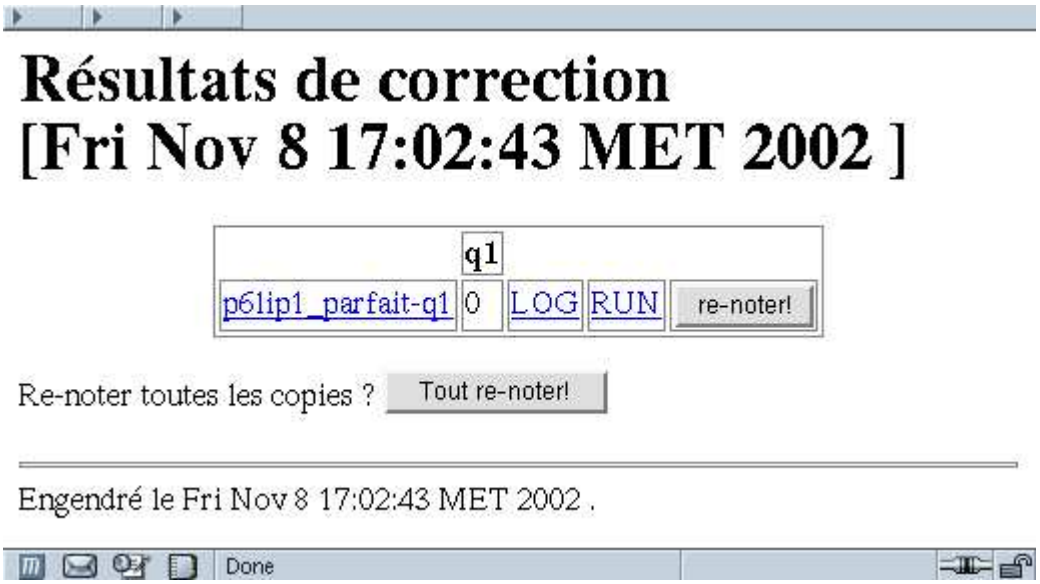


FIG. 1 – Petit serveur de notation

   Il n'y a pas de problèmes de sécurité ici car vous ne tournez que votre code à vous, celui du CFSkit en qui vous avez (bien sûr) confiance et les pseudo-copies que vous avez vous-même écrites. Il ne faut surtout pas faire la même chose avec les vraies copies des vrais étudiants (cf. section 4 plus bas).

   Créez une entrée par défaut dans votre *Makefile* global pour automatiser tout cela. De nombreux paramètres existent et sont décrits dans les fichiers *src/*.mkf*.

7. Améliorez l'énoncé, les pseudo-copies, les tests jusqu'à satisfaction.

3.3 Base de données

Une base de données n'est pas nécessaire, c'est seulement utile pour archiver des notes et les présenter via des pages dynamiques (PHP ou autres) exploitant cette base. Les deux derniers points de la précédente énumération requéraient un serveur Postgresql et le droit de créer une nouvelle base. Les scripts créent alors, d'eux-mêmes, la base, les trois tables dans cette base et leur contenu.

4 Déploiement

Le déploiement de CFSkit, c'est-à-dire la notation des copies des étudiants avec leur code potentiellement malveillant mais fort probablement bourré d'erreurs, est pour le moins délicat. Le premier déploiement, décrit plus haut, permet de noter les pseudo-copies avec une base locale sans entrer dans les difficultés listées ci-dessous. Dans un vrai déploiement, visant à noter des copies d'étudiants se trouvant sur un serveur de l'université avec une machine de confinement différente et une base de données sur un troisième serveur, avec des résultats qui doivent migrer sur un quatrième serveur HTTP via des formulaires PHP à placer quelque part, il y a de très nombreux paramètres à asujettir de façon cohérente.

4.1 Déploiement personnel

Si vous ne souhaitez qu'une sécurité faible ou que vous avez confiance en vos étudiants, voici un déploiement simple et utile lorsque vous n'avez pas accès aux serveurs de l'université par exemple parce que vous corrigez vos copies, non connecté, pendant vos vacances d'hiver à Acapulco. Ce déploiement limite un peu les dégâts possibles puisque le code des étudiants sera tourné avec l'identité d'un utilisateur distinct de vous mais il n'empêchera pas que le code tourné fasse accès au réseau (envoi de courrier, réception de programme, etc.), ou bousille les copies en cours de correction simultanée (puisque toutes sont tournées avec la même identité). Vous n'avez que peu de moyens vous permettant de vérifier si ce fut le cas.

On supposera dans ce qui suit que *MonExam/* est le chemin menant au répertoire contenant les fichiers relatifs au contrôle à corriger. Ce qui suit n'est qu'une description de haut niveau de ce qu'il faut faire. Une trace plus commentée (prise dans un autre contexte) apparaît en section 5.1 ainsi qu'un exemple sur une épreuve réelle en section D.

1. Créer un nouvel utilisateur, disons `MonExam`, sur votre machine.
2. En tant que ce nouvel utilisateur, créer un répertoire *public_html* afin de servir des pages qui seront accessibles via le serveur `httpd` de votre machine.
3. En tant que `root`, installez les copies à corriger dans le répertoire *public_html/students*. Soyez sûr des droits afin qu'elles soient inaltérables (au minimum non modifiables par l'utilisateur `MonExam`). Un simple lien vers votre *MonExam/students/* peut bien sûr suffire.
4. En tant que `root`, créez le répertoire *public_html/tests* contenant les tests. Soyez sûr des droits afin qu'ils soient inaltérables. Un simple lien vers votre *MonExam/tests/* peut bien sûr suffire.
5. En tant que `root`, installez les sources du CFSkit dans *public_html/src*. Soyez sûr des droits afin que ces fichiers soient inaltérables. Un simple lien vers votre *MonExam/src/* peut bien sûr suffire. Dans le répertoire *public_html/src*, vous devez trouver, par exemple, le fichier `run.mkf`.
6. En tant que `root`, créez un fichier *public_html/Makefile* incluant *src/local.mkf* du CFSkit définissant tous les paramètres de correction.
   Généralement je fais un lien vers *MonExam/Makefile* où je mets des tas de choses. Voici un exemple de *Makefile* minimal (ça doit être à peu près cela) :

```
work : local.mark.all.students.tgz \
       local.rebuild.db.on.other
EXAMINATION     =       MonExam
LOCAL_USER_DIR  =       /home/MonExam
LOCAL_TESTS_DIR =       ${LOCAL_USER_DIR}/public_html/tests/
LOCAL_TGZDIR    =       ${LOCAL_USER_DIR}/public_html/students/
LOCAL_RUN_DIR   =       ${LOCAL_USER_DIR}/public_html/tmp/
LOCAL_LOG_DIR   =       ${LOCAL_USER_DIR}/public_html/logs/
OTHER_DB        =       ${EXAMINATION}
OTHER_DBPREFIX  =       ${EXAMINATION}
```

```
CFSKIT_DIR      =       ${LOCAL_USER_DIR}/public_html/src
include ${CFSKIT_DIR}/local.mkf
include ${CFSKIT_DIR}/mark.mkf
```

7. Créez le répertoire *public_html/logs* écrivable par `MonExam`. Ce répertoire contiendra les copies corrigées. Il sera ainsi possible de parcourir les copies corrigées avec l'URL http://localhost/~MonExam/logs/.
8. En tant qu'utilisateur `MonExam` copiez (ou liez) les pages *public_html/src/php/index.php* et *public_html/src/php/libcfskit.php* en *public_html/* et modifiez la page *index.php* à votre convenance (voir annexe B. Il y a quelques variables à assujettir en début de fichier. Cette page vous servira à regarder les résultats de notation (notes et copies) par extraction des notes de la base (cf. ci-dessous) et ainsi vous permettre de mettre au point définitivement les programmes de notation (il s'agit en général de prendre en considération des anomalies couramment observées probablement dues à une imprécision ou ambiguïté dans l'énoncé).
   Généralement je fais un lien vers *MonExam/site/index.php* (après modification) et vers *MonExam/src/php/libcfskit.php*.
   Cette page de garde sera accessible avec l'URL http://localhost/~MonExam/.
9. Créez le répertoire *public_html/tmp* écrivable par `MonExam`. Ce répertoire contiendra les répertoires utilisés pour corriger les copies.
10. Les résultats étant stockés dans une base de données Postgresql dont le nom est indiqué dans la variable `OTHER_DB` du *Makefile* précédent, il faut créer cette base.
11. Incarnez l'utilisateur `MonExam` et tournez la commande suivante qui devrait corriger toutes les copies, à condition d'avoir bien paramétré le Makefile. Du fait de la séparation des utilisateurs sous Unix, ni les copies, ni les tests, ne devraient être modifiés.

```
cd public_html ; make local.mark
```

Pour présenter les notes et les copies, il faut :

1. les pages *public_html/*.php* vers le serveur devant les héberger,
2. les copies des étudiants en *public_html/students/*
3. les copies corrigées en *public_html/logs/*
4. les trois tables de la base (préfixées par `MonExam`) vers une base de données accessible du serveur HTTP devant héberger les pages.

La règle `install.results` permet cela à condition de la paramétrer; voir ces paramètres dans le fichier *local.mkf*.

4.2 Base de données

Une base de données est nécessaire pour archiver des notes et les présenter via les pages dynamiques fournies dans le CFSkit. Il faut pour cela avoir accès à une base Postgresql, pouvoir y créer les trois tables contenant les notes des différentes questions (ainsi que certains faits statistiques). Afin de présenter les résultats, la base de données doit contenir une table Login et une table Person permettant de retrouver les noms et prénoms des étudiants à partir de leur nom de connexion (*login*).

Lorsque satisfait de la notation, il vous faut aménager un accès aux résultats aux étudiants. On peut pour cela utiliser Delivery_Builder comme indiqué ci-après ou transporter les résultats de notation sur un serveur HTTP comme indiqué plus haut.

4.3 Delivery_Builder

Fabrice Kordon a encapsulé CFSkit dans son outil Delivery_Builder et a donc paramétré le CFSkit relativement à la configuration de l'UFR d'informatique de l'UPMC. Tous les détails sur le site[3] de Delivery_Builder.

[3] `http://www.infop6.jussieu.fr/licence/2002/delivery_builder/`

Ce déploiement utilise mygale mais est indépendant des bases de données de l'UFR.
Si vous souhaitez adopter le déploiement personnel à partir de Delivery_Builder, voici comment faire :

1. récupérer les copies à partir de Delivery_Builder. Pour cela, aller sur le site de ressources de Delivery_Builder puis dans la partie réservée aux enseignants, choisir le bouton ListeAdminModule, choisir au sein du module concerné le bouton AccederAdministrationGroupeTD/TP puis choisir une livraison et cliquer sur le bouton etatLivraisons enfin importerArchiveUnique et sur la page résultante le lien. Attention, vous n'avez que cinq minutes pour récupérer ce fichier `tgz` !
2. vous pouvez alors retourner à la section 4.1.

4.4 Déploiement sur une machine de confinement

Afin d'éviter que les copies des étudiants ne viennent perturber votre machine ou la correction en cours, il vaut mieux les exécuter sur une machine dite de confinement sans démon, sans droit mais identique à celles sur lesquelles les étudiants ont composé. À l'UFR d'informatique de l'UPMC, cette machine se nomme mygale. Ce déploiement est propre à l'UFR et se réfère à son organisation ouèbe actuelle.

La difficulté de ce déploiement est que au moins trois machines sont impliquées : le serveur ouèbe et base de données papillon (inaccessible par `sh`), la machine de confinement mygale (accessible seulement par `ssh` et ne voyant pas papillon et ses ressources) enfin la machine centrale tique qui permet de commander les deux autres. N'oublions pas non plus la machine personnelle de l'enseignant sur laquelle se trouve les originaux de ses tests, pages et autres programmes. Tout n'est pas accessible de partout et tout n'a pas le même nom suivant le point de vue !

1. Supposons que, sur tique, le répertoire qui servira à présenter les résultats aux étudiants est */Ens/licence/2002/cct/cfs3/*, l'url associée sera servie par papillon comme : http://www.infop6.jussieu.fr/licence/2002/cct/cfs3/. Vous pouvez, pendant que vous faites tout ce qui suit, protéger ce répertoire des regards indiscrets en y plaçant un fichier *.htaccess*.
2. Supposons qu'un compte libre sur mygale soit test87. Vous pouvez changer son mot de passe afin d'empêcher un autre enseignant d'utiliser ce même compte pendant que vous y travaillez. Vous pouvez également créer des clés `ssh` pour vous simplifier encore plus la vie plus tard (mais vous la compliquez en attendant).
3. supposons que les copies des étudiants archivées par Delivery_Builder soient en */Ens/licence/2002/livraisons/LIC_CFS2_27_05_0*
4. Entre la machine de confinement et tique, il n'y a que le répertoire */Infos/licence/2002/cfs3* de commun. C'est par ce répertoire que vont passer les copies, les tests et la dernière version du CFSkit, de tique à mygale et, en plus, l'utilisateur test87 de mygale ne pourra que les lire pas les altérer !

   Il faut bien sûr créer ce répertoire */Infos/licence/2002/cfs3*.

   Attention, ne confondez pas les répertoires publics (car visibles depuis Internet) préfixés par */Ens/* et ceux préfixés par */Infos/* qui ne sont communs qu'à tique et mygale.
5. Copiez vos tests depuis votre machine personnelle sur */Ens/licence/2002/cfs3/tests/*. Ce répertoire contient normalement une série de répertoires *Q1*, etc. Ces tests sont à une position où ils pourront être rendus visibles des étudiants (ce qui leur donne confiance sur la répétabilité des tests).
6. Créez */Infos/licence/2002/cct/cfs3/Makefile* comme suit. Attention, le répertoire */Infos* est visible de tous les étudiants, n'y mettez pas d'information sensible (En fait, il faudrait avoir une partition commune entre tique et mygale qui ne soit pas visible des étudiants, cela serait plus sûr ! FUTURE)

   Ce *Makefile* fixe les variables du fichier *src/mark.mkf* du CFSkit. Les commentaires ont tous été retirés mais vous devriez y retrouver les informations données plus haut.

```
work :
        @echo Tout va bien

# Variables de mark.mkf
# ############## On WEBSERVER: ### papillon
```

```
SERVER_RESULTS_URL=/licence/2002/cct/cfs3
# ############# On OTHER: ### tique
EXAMINATION=cfs3
RT_RUN_FLAGS=-n
CONFINER_USER=test87
OTHER_ORG_TGZDIR=/Ens/licence/2002/livraisons/LIC_CFS2_27_05_03AM
OTHER_TRANSIT_DIR=/Infos/licence/2002/cfs3
OTHER_RESULTS_DIR=/Ens${SERVER_RESULTS_URL}
OTHER_ORG_CFSKIT_DIR=/Ens/licence/auth/private/delbul/cfskit/src
OTHER_ORG_TESTS_DIR=/Ens${SERVER_RESULTS_URL}/tests
OTHER_DB=nlicence -h papillon
OTHER_DBPREFIX=${EXAMINATION}
# ############# On CONFINER: ### mygale
CONFINER_RESULTS_DIR=/home/${CONFINER_USER}/logs
CONFINER_CFSKIT_DIR=${OTHER_TRANSIT_DIR}/src
CONFINER_TGZDIR=${OTHER_TRANSIT_DIR}/students
CONFINER_TESTS_DIR=${OTHER_TRANSIT_DIR}/tests
CONFINER=mygale.infop6.jussieu.fr
CONFINER_MAKEFILE=${OTHER_TRANSIT_DIR}/Makefile

ifeq (${CONFINER_CFSKIT_DIR}/mark.mkf, $(wildcard ${CONFINER_CFSKIT_DIR}/mark.mkf))
include ${CONFINER_CFSKIT_DIR}/mark.mkf
else
include ${OTHER_ORG_CFSKIT_DIR}/mark.mkf
endif
```

7. Tentez de tourner la commande suivante qui va tenter d'installer les copies à corriger, le CFSkit et les tests dans le répertoire */Infos/licence/2002/cfs3* afin qu'ils soient visibles de la machine de confinement. Si tout se passe bien cette commande ne doit être effectuée qu'une seule fois (vous devrez néanmoins la recommencer si vous changez vos tests et ceci impliquera probablement votre machine personnelle ou si le CFSkit évolue pendant que vous corrigez).

   ```
   tique% cd /Infos/licence/2002/cfs3
   tique% make before.marking.on.other
   ```

8. Maintenant que la machine de confinement peut trouver les copies, les tests ainsi que la dernière version du CFSkit via le répertoire (lisible mais non modifiable pour la machine de confinement) */Infos/licence/2002/cfs3*, il faut tourner la commande suivante pour lancer la notation des copies. Cette commande va vous demander le mot de passe pour aller sur mygale puis va noter, pendant un certain temps les copies, une à une, test par test, ...

   ```
   tique% make marking.on.other
   ```

9. Pendant que la commande précédente tourne, vous pouvez obtenir des statistiques partielles et instantanées avec la commande suivante. Vous aurez à donner un mot de passe si vous n'avez pas créé de clés `ssh`.

   ```
   tique% make after.marking.on.other
   ```

   Toutes ces notes sont mises, par défaut, dans la base de données de l'UFR. Les tables qui seront créées vous appartiendront.

10. Lorsque toutes les corrections sont achevées, vous pouvez rapatrier les résultats depuis la machine de confinement avec la même commande que précédemment. Il ne reste plus qu'à mettre à jour la page *index.php* pour que les étudiants puissent accéder à leurs résultats. Consultez l'annexe B pour voir les paramètres à modifier.
11. Publiez (par une nouvelle fraîche) l'URL de consultation des copies auprès des étudiants. Cette URL devrait être http://www.infop6.jussieu.fr/licence/2002/cct/cfs3/. Retirez le fichier *.htaccess* si vous en avez mis un.

5 La distribution du CFSkit

Cette section décrit très rapidement le contenu de la distribution.

La distribution contient quelques documentations dans le répertoire *doc* : cette documentation et un article (en français) sur les premières leçons tirées de cette expéricence (présenté à la conférence TICE 2002).

Le répertoire *src* contient les sources (bash, C, Perl et Makefile) du CFSkit. Ils sont tous distribués sous licence GPL[4].

Le sous-répertoire *src/php* contient quelques pages PHP permettant d'arpenter les résultats d'une notation en masse.

Le sous-répertoire *src/tests* contient quelques tests internes à la distribution suivant eux-mêmes les principes du CFSkit.

5.1 Exemple

Le répertoire *TME* est un petit exemple d'examen de notation en masse : il contient un énoncé, des tests, des pseudo-copies et des copies (les copies nommées omega et upsilon bouclent intentionnellement mais finissent par être corrigées !). Vous pouvez vous faire la main en écrivant de nouveaux tests pour la première question.

Voici un petit mode d'emploi pour tourner cet examen sur sa propre machine Linux. Cette section utilise des pages PHP qui n'ont pas été décrites précédemment. Elle utilise la méthode de déploiement personnel vue en section 4.1.

Ouvrez une fenêtre pour devenir tout puissant (root, quoi !) sur votre machine. Créez alors un utilisateur nommé tme sur votre machine et son répertoire *public_html/*. Il y a des utilitaires comme `useradd tme` qui doivent faire ce travail.

```
root% useradd tme
root% mkdir -p ~tme/public_html
```

En tant que root, installez comme suit les fichiers de cette archive (on les installe en tant que root afin d'éviter leur corruption par l'utilisateur tme). Dans ce qui suit, la variable `TME` est supposée contenir le chemin vers le répertoire TME décrivant l'examen qu'il s'agit de corriger. Ce répertoire contient les répertoires standard : *students* vers les copies à corriger, *tests* vers les procédures de notation, etc.

```
root% cd ~tme/public_html
root% ln -s $TME/students .
root% ln -s $TME/tests .
root% cp $TME/tme.mkf Makefile
root% cp src/php/*.php .
root% chown -R tme.tme $TME/pseudos
root% chown -R tme.tme Makefile *.php
root% chown tme.tme .
```

[4] `http://www.fsf.org/copyleft/gpl.html`

Il faut maintenant inscrire l'utilisateur tme comme utilisateur de postgres pouvant créer des bases.

```
root% su - postgres
postgres% createuser tme
Shall the new user be allowed to create databases? (y/n) y
Shall the new user be allowed to create more new users? (y/n) n
CREATE USER
postgres% exit
```

Incarnez l'utilisateur tme :

```
root% su - tme
tme% cd public_html
tme% ls
index.php      libcfskit.php@      logs/
Makefile@      src@                students@
tests@         tmp/
```

Enfin l'on peut noter les pseudo-copies (six sont prédéfinies). Les copies nommées omega et upsilon bouclent intentionnellement mais finissent par être corrigées.

```
tme% make pseudos.mark
*****************************************
CFSKIT $Revision: 1.37 $
*****************************************
*****************************************
Prepare all tests scripts if needed ...
*****************************************
cd tests && for m in * ; do if [ -d $m ] ; then \
        (cd $m && if [ -f Makefile ] ; then make ; fi ) ; fi ; done
*****************************************
Prepare all pseudos if needed ...
*****************************************
cd pseudos && for m in * ; do if [ -d $m ] ; then \
        (cd $m && if [ -f Makefile ] ; then make ; fi ) ; fi ; done
cd pseudos && for d in * ; do \
if [ -d $d ] ; then ( cd $d && tar czhf ../$d.tar.gz .) ; fi ; done
*****************************************
Here are the pseudos to mark ...
*****************************************
ls -l pseudos/p6lip*.tar.gz
-rw-rw-r-- 1 tme tme  375 Jul 5 15:06 pseudos/p6lip101_NOBODY_bc.tar.gz
-rw-rw-r-- 1 tme tme  446 Jul 5 15:06 pseudos/p6lip102_NOBODY_perl.tar.gz
-rw-rw-r-- 1 tme tme  573 Jul 5 15:06 pseudos/p6lip103_NOBODY_perfect.tar.gz
-rw-rw-r-- 1 tme tme  162 Jul 5 15:06 pseudos/p6lip104_NOBODY_nul.tar.gz
-rw-rw-r-- 1 tme tme  257 Jul 5 15:06 pseudos/p6lip105_NOBODY_omega.tar.gz
-rw-rw-r-- 1 tme tme  263 Jul 5 15:06 pseudos/p6lip106_NOBODY_upsilon.tar.gz

*****************************************
```

```
Preparing directories ...
*****************************************
rm -rf /home/tme/public_html/tme2002Jul01/logs
mkdir -p /home/tme/public_html/tme2002Jul01/logs
rm -rf /home/tme/public_html/tme2002Jul01/tmp
mkdir -p /home/tme/public_html/tme2002Jul01/tmp
*****************************************
Starting to mark all pseudos ...
*****************************************
export CFSKIT_DIR=‘cd /home/tme/public_html/src; pwd‘ ;\
export RT_KEEP_RUN_DIR=true ;\
export RT_TGZ_DIR=‘pwd‘/pseudos ;\
export RT_TESTS_DIR=‘pwd‘/tests ;\
export RT_MARK_DIR=$CFSKIT_DIR ;\
export RT_LINK_TEST=$RT_MARK_DIR/genlink.sh ;\
export RT_HTTP_PREFIX=file://‘pwd‘ ;\
export RT_SESSION=tme2002Jul01 ;\
export RT_LOG_DIR=/home/tme/public_html/tme2002Jul01/logs ;\
cd /home/tme/public_html/tme2002Jul01/tmp && time bash $RT_MARK_DIR/run.sh $RT_TGZ_DIR
Marking directory /home/tme/public_html/pseudos ...
Marking /home/tme/public_html/pseudos/p6lip101_NOBODY_bc.tar.gz ...
Somme des points de la question Q1: 1.
Somme des points de la question Q2: 1.
Total obtenu: 2
*** Directory /home/tme/public_html/pseudos, copie 1/6.
Marking /home/tme/public_html/pseudos/p6lip102_NOBODY_perl.tar.gz ...
Somme des points de la question Q1: 0.
Somme des points de la question Q2: 0.
Total obtenu: 0
*** Directory /home/tme/public_html/pseudos, copie 2/6.
Marking /home/tme/public_html/pseudos/p6lip103_NOBODY_perfect.tar.gz ...
Somme des points de la question Q1: 1.
Somme des points de la question Q2: 1.
Total obtenu: 2
*** Directory /home/tme/public_html/pseudos, copie 3/6.
Marking /home/tme/public_html/pseudos/p6lip104_NOBODY_nul.tar.gz ...
Somme des points de la question Q1: 0.
Somme des points de la question Q2: 0.
Total obtenu: 0
*** Directory /home/tme/public_html/pseudos, copie 4/6.
Marking /home/tme/public_html/pseudos/p6lip105_NOBODY_omega.tar.gz ...
Somme des points de la question Q1: 0.1.
Somme des points de la question Q2: 0.1.
Total obtenu: 0.2
*** Directory /home/tme/public_html/pseudos, copie 5/6.
Marking /home/tme/public_html/pseudos/p6lip106_NOBODY_upsilon.tar.gz ...
Somme des points de la question Q1: 1.
Somme des points de la question Q2: 1.
Total obtenu: 2
*** Directory /home/tme/public_html/pseudos, copie 6/6.
41.41user 2.47system 0:45.11elapsed 97%CPU (0avgtext+0avgdata 0maxresident)k
```

```
0inputs+0outputs (68008major+44399minor)pagefaults 0swaps
*****************************************
End of marking process.
*****************************************
cd pseudos && rm *.tar.gz
make compute.db.stats
make[1]: Entering directory ‘/home/tme/public_html’
*****************************************
Computing some statistics ...
*****************************************
dropdb tmptme2002Jul01
ERROR:  DROP DATABASE: database "tmptme2002Jul01" does not exist
dropdb: database removal failed
make[1]: [compute.db.stats] Error 1 (ignored)
export CFSKIT_DIR=‘cd /home/tme/public_html/src; pwd‘ ;\
export LOCAL_DB="tmptme2002Jul01" ;\
export LOCAL_DBPREFIX="tme2002Jul01" ;\
export LOCAL_DBTABLES="tme2002Jul01_student tme2002Jul01_mark tme2002Jul01_fact" ; \
export LOCAL_DBFLAGS="--set STUDENT=tme2002Jul01_student --set MARK=tme2002Jul01_mark --
export RT_LOG_DIR="/home/tme/public_html/tme2002Jul01/logs" ; \
bash $CFSKIT_DIR/test-stat.sh
CREATE DATABASE
CREATE
REVOKE
GRANT
GRANT
CREATE
REVOKE
GRANT
GRANT
CREATE
REVOKE
GRANT
GRANT
CREATE
INSERT 404700 1
INSERT 404701 1
INSERT 404702 1
INSERT 404703 1
INSERT 404704 1
INSERT 404705 1
insertion des notes et faits
INSERT 404706 1
INSERT 404707 1
INSERT 404708 1
INSERT 404709 1
INSERT 404710 1
INSERT 404711 1
INSERT 404712 1
INSERT 404713 1
INSERT 404714 1
```

```
INSERT 404715 1
INSERT 404716 1
INSERT 404717 1
INSERT 404718 1
INSERT 404719 1
INSERT 404720 1
INSERT 404721 1
INSERT 404722 1
INSERT 404723 1
CREATE
INSERT 0 6
UPDATE 6
UPDATE 6
   sid    | q1  | q2
----------+-----+-----
 p6lip101 |   1 |   1
 p6lip102 |   0 |   0
 p6lip103 |   1 |   1
 p6lip104 |   0 |   0
 p6lip105 | 0.1 | 0.1
 p6lip106 |   1 |   1
(6 rows)


********** Min, moyenne et max par session et par question:
   session    | question | count | min |        avg        | max
--------------+----------+-------+-----+-------------------+-----
 tme2002Jul01 | Q1       |     6 |   0 | 0.516666666666667 |   1
 tme2002Jul01 | Q2       |     6 |   0 | 0.516666666666667 |   1
(2 rows)

SELECT
SELECT
********** Min, moyenne et max par notes d etudiant par session:
SELECT
   session    |  count   | min |       avg        | max
--------------+----------+-----+------------------+-----
 tme2002Jul01 | 6 copies |   0 | 1.03333333333333 |   2
(1 row)

make[1]: Leaving directory ‘/home/tme/public_html’
make compute.logs.index.html
make[1]: Entering directory ‘/home/tme/public_html’
****************************************
Synthetizing results ...
****************************************
export CFSKIT_DIR=‘cd /home/tme/public_html/src; pwd‘ ;\
cd /home/tme/public_html/tme2002Jul01/logs && \
export RT_RUN_DIR="/home/tme/public_html/tme2002Jul01/tmp" ;\
export RT_LOG_DIR="/home/tme/public_html/tme2002Jul01/logs" ; \
export LOCAL_DB="tmptme2002Jul01" ;\
```

```
export LOCAL_DBPREFIX="tme2002Jul01" ;\
perl $CFSKIT_DIR/generate-index.pl > index.html
make[1]: Leaving directory '/home/tme/public_html'
****************************************
CFSKIT $Revision: 1.37 $
****************************************
```

L'affichage final montre les moyennes obtenues par question. Pour consulter les résultats, vous pouvez consulter, par exemple avec `pgaccess`, la base tmptme2002Jul01 mais il y a plus simple : exécutez :

```
tme% make x.run.re-mark.server
```

et dirigez votre navigateur vers l'url[5]. L'affichage et les boutons parlent d'eux-mêmes. Ce serveur ne répond qu'aux requêtes provenant de votre machine (pour des raisons de sécurité car il est capable de servir tout fichier visible de l'utilisateur tme).

Pour noter les vraies copies (il y en a 17 dont les six pseudo-copies précédentes), il faut exécuter ce qui suit :

```
tme% make local.mark
```

Pour pouvoir consulter ces résultats, il faut diriger votre navigateur vers l'URL[6] (à condition qu'un serveur `httpd` tourne sur votre machine et qu'elle permette l'accès aux répertoires *public_html* des utilisateurs). Il faut bien sûr aussi avoir mis à jour la page *index.php* et posséder les tables des étudiants et de leur nom de connexion (*login*).

6 Conclusions

Pas de bogues connues !

Références

[QC02] Christian Queinnec and Emmanuel Chailloux. Une expérience de notation en masse. In *TICE 2002 – Technologies de l'Information et de la Communication dans les Enseignements d'Ingénieurs et dans l'industrie – Conférences ateliers*, pages 403–404, Lyon (France), November 2002. Institut National des Sciences Appliquées de Lyon. version complète disponible en `http ://www.infop6.jussieu.fr/licence/2001/cct/cfsreport.ps.gz`.

A Style

Un style *cfs.sty* existe pour LaTeX. Son adaptation pour HeVeA existe aussi, nommée *cfs.hva*, afin d'engendrer de l'HTML à partir de LaTeX. Je suis pour l'instant l'unique utilisateur de ce fichier de style qui n'est peut-être pas si indépendant que cela de mon environnement de travail. Signalez-moi les problèmes !

La structure du corps d'un fichier d'examen est alors la suivante :

```
\begin{question}
Écrire la fonction qui ...
```

[5] `http://127.0.0.1:56001/home/tme/public_html/tme2002Jul01/logs/index.html`
[6] `http://127.0.0.1/~tme/`

```
\begin{livraison}{2002nov}
\item le fichier ...
\end{livraison}

\begin{notation}{4}
\note{1} si votre programme ...
\bonus{1} si votre programme est dans les 10% les plus rapides.
   \assertion Les programmes seront invoqués avec des arguments corrects.
\end{notation}

\end{question}
```

L'environnement `question` enchâsse la définition d'une question. La question commence par l'énoncé se continue par la description (une énumération) des fichiers à rendre via Delivery_Builder (matérialisé par l'environnement `livraison`) et s'achève par une description des tests qui seront opérés (l'environnement `notation`).

L'environnement `livraison` prend un argument qui est l'indicateur de livraison pour Delivery_Builder. Il est pratique de n'avoir qu'un unique indicateur de livraison par examen ou épreuve (il est pénible d'en avoir un par question).

L'environnement `notation` prend un argument qui est le total des points attribués à la question. Le contenu de cet environnement est une liste dont les items sont des `note`s ou des `bonus`. Ces deux macros prennent un unique argument qui est le nombre de points que peut rapporter le test. La somme des points des `note`s doit être égale à l'argument de la `notation`. Comme leur nom l'indique, les `bonus` viennent en supplément.

Pour la citation de code, j'utilise `l2t` et l'environnement `snippet`. Pour différencier (en HTML et dvi) les entrées de l'ordinateur de celle de l'humain, j'utilise les macros `SwitchToUserInputFont` et `SwitchToComputerFont`.

B Page PHP

Dans le répertoire *src/php/* se trouvent une page PHP nommée *index.php* et sa bibliothèque associée *libcfskit.php*. Le début de la page *index.php* doit être adaptée aux conditions de l'examen afin de pouvoir présenter les notes et les copies corrigées. Cette page doit être placée sur le serveur ouèbe. Voici les paramètres :

css l'URL (relative ou absolue) de la feuille de style associée à la page *index.php*.

email le courriel à faire apparaître en cas de remarque sur la page. C'est notamment ce courriel qui servira aux étudiants pour contester leur note.

db_name le nom de la base de données à lire.

db_user le nom de l'utilisateur de la base de données à utiliser pour se connecter.

db_password le mot de passe de l'utilisateur de la base de données à utiliser pour se connecter.

tgzPrefix le répertoire où trouver les copies non corrigées des étudiants. La plupart du temps c'est le sous-répertoire *students* qui est un lien symbolique vers le répertoire rempli par Delivery_Builder (quelque chose sur tique, comme un sous-répertoire de */Ens/licence/2002/livraisons/*.

uuidPrefix le répertoire où trouver les copies corrigées des étudiants. Ce répertoire est souvent nommé *log* et contient des sous-répertoires nommés avec un UUID qui contiennent des pages `all.html`.

imagePrefix est une URL menant à la photo de l'étudiant. Ce paramètre n'est pas obligatoire.

title1 le titre de la page présentant les notes de tous les étudiants.

title2 le titre de la page présentant les notes question par question obtenue par un étudiant sur toutes ses copies corrigées (souvent seulement la dernière).

À ces paramètres, il faut ajouter une fonction `compute_total` qui prend le tableau des notes obtenues par une copie question par question et qui calcule la note finale (souvent une simple addition mais qui peut être complétée par une homothétie, une translation, une troncature, etc.)

C Java

Pour tester des classes Java, j'ai créé ce paquetage `fr.lip6.qnc.cfskit` mettant en œuvre Junit[7]. On écrit les tests comme en XP (*eXtreme Programming*) et on compte simplement les tests qui marchent et ceux qui ne marchent pas. Ces nombres peuvent ensuite être repris dans un test shell pour fabriquer une note.

Contrairement à JUnit qui ne compte que le nombre de tests ayant fonctionné, le paquetage du cfskit compte aussi le nombre d'assertions ayant fonctionné ce qui donne une meilleure idée de la difficulté de ce qui était demandé.

Voici un exemple de bibliothèque mettant en œuvre ces classes. Tout d'abord un test en Java (il y en a 4 autres testant d'autres points ou d'autres méthodes demandées) :

```
// $Id$

import junit.framework.*;
import fr.upmc.infop6.mlo.*;

public class Test_Option1 extends fr.lip6.qnc.cfskit.TestCase {

  public void test_new ()
    throws Exception {

    // des noms d'options simples:

    Option o1 = new Option("--a");
    assertNotNull(o1);
    assertEquals(o1.getName(), "a");

    Option o11 = new Option("--aaldjfaldf");
    assertNotNull(o11);
    assertEquals(o11.getName(), "aaldjfaldf");
  }

}

// end of Test_Option1.java
```

Puis quelques fonctions shell pour mettre en œuvre Java :

```
#! /bin/sh

# Ce script procure deux fonctions standard pour le test de classes
# Java.  La premiere compile la classe et distribue quelques dixiemes
# de point aux differentes phases reussies de compilation. La seconde
# lance les tests associes a la question et distribue les points
# suivant le nombre de tests.
```

[7] `http://www.junit.org/`

```
# Ces fonctions ont ete utilisees pour corriger
# le partiel de MLO en decembre 2002,
# le rattrapage de MLO en septembre 2002.

# Charger des constantes additionnelles:

if [ -f ${RT_TESTS_DIR}/globals.sh ]
then source ${RT_TESTS_DIR}/globals.sh
fi

# GAIN le gain total pour la compilation de cette classe.
# JAVAFILE le fichier java a compiler (ce qui peut induire que d'autres
#          fichiers pourront etre aussi compiles).
# Le GAIN est saucissone en 1/10 pour l'existence du fichier, 4/10 si
# aucune erreur de compilation n'est percue et 5/10 pour l'existence
# du fichier .class resultant a condition que la classe soit publique.

function rt_compile_class {
    local GAIN=$1
    local JAVAFILE=$2
    rt_exist $JAVAFILE
    rt_win $GAIN/10

    rt_echo Voici votre classe '<tt>'$JAVAFILE'</tt>'
    rt_file_echo $JAVAFILE
    cmd="${JAVAC} $JAVAFILE"
    rt_echo Je compile votre classe '<tt>'$JAVAFILE'</tt>' \
            avec la commande '<tt>'$cmd'</tt>'.
    $cmd     2>${RT_TMP}/cerr
    if [ -s ${RT_TMP}/cerr ]
    then
        rt_error La compilation a produit ces avertissements ou erreurs:
        rt_start_stderr
        cat ${RT_TMP}/cerr
        rt_stop_stderr
        rm ${RT_TMP}/cerr
    else
        rt_echo "Aucun avertissement ni erreur, bravo!"
        rt_win '4 * '${GAIN}/10
    fi

    rt_exist ${JAVAFILE%%.java}.class

    # La classe est-elle publique ?
    cmdp="${JAVAP} ${JAVAFILE%%.java}"
    $cmdp > ${RT_TMP}/cl
    #rt_warning "$cmdp produit: "                                #DEBUG
    #rt_file_echo ${RT_TMP}/cl                                   #DEBUG
    if grep -q 'public class' < ${RT_TMP}/cl > /dev/null
    then
        rt_win ${GAIN}/2
```

```
    else
        rt_warning "Votre classe n'est pas publique! Les tests ne pourront " \
            "donc pas fonctionner correctement (relisez les consignes " \
            "g&eacute;n&eacute;rales de l'&eacute;nonc&eacute;). " \
            "Je tente de corriger (d'ajouter <tt>public</tt> devant " \
            "<tt>class</tt>). "
        # On preserve au cas ou:
        mv ${JAVAFILE}              ${JAVAFILE}.ORG
        mv ${JAVAFILE%%.java}.class ${JAVAFILE%%.java}.class.ORG
        sed -e 's/class/public class/' < ${JAVAFILE}.ORG > ${JAVAFILE}
        rt_warning "Voici votre classe modifi&eacute;e:" '<tt>'$JAVAFILE'</tt>'
        rt_file_echo $JAVAFILE
        rt_warning "Je compile votre classe modifi&eacute;e " \
            '<tt>'$JAVAFILE'</tt>' "avec la commande" '<tt>'$cmd'</tt>'.
        $cmd     2>${RT_TMP}/cerr
        if [ -s ${RT_TMP}/cerr ]
        then
            rt_error La compilation a produit ces avertissements ou erreurs:
            rt_start_stderr
            cat ${RT_TMP}/cerr
            rt_stop_stderr
            rm ${RT_TMP}/cerr
            rt_warning "Je reprend donc vos classes originales. "
            mv ${JAVAFILE}.ORG              ${JAVAFILE}
            mv ${JAVAFILE%%.java}.class.ORG ${JAVAFILE%%.java}.class
            touch ${JAVAFILE%%.java}.class
        else
            $cmdp > ${RT_TMP}/cl
            if grep -q 'public class' < ${RT_TMP}/cl > /dev/null
            then
                rt_warning "Cela semble repar&eacute;, je continue. "
            else
                rt_warning "La r&eacute;paration semble poser " \
                    "probl&egrave;me, je restaure vos classes originales. "
                mv ${JAVAFILE}.ORG              ${JAVAFILE}
                mv ${JAVAFILE%%.java}.class.ORG ${JAVAFILE%%.java}.class
                touch ${JAVAFILE%%.java}.class
            fi
        fi
    fi
    rm -f ${RT_TMP}/cl
    # fin du test de classe publique

    rt_exist ${JAVAFILE%%.java}.class
}


# TOTAL le gain total et maximal attendu
# COMPILE_GAIN par compilation reussie d'un Test_*.java
# EXECUTE_GAIN par methode test_* reussie
# ASSERTION_GAIN par assertion reussie
```

```
function rt_test_with_classes {
    local TOTAL=$1
    local COMPILE_GAIN=$2
    local EXECUTE_GAIN=$3
    local ASSERTION_GAIN=$4

    local NCOMPIL=‘ls ${RT_TESTS_DIR}/${RT_Q}/Test_*.java | wc -l‘
    local NTESTS=‘egrep ’public *void *test’ ${RT_TESTS_DIR}/${RT_Q}/Test_*.java | wc -l
    # On utilise JUnit
    rt_echo "<p> J’installe mes" $NCOMPIL "classes contenant" $NTESTS "tests" \
        "en tout. Elles utilisent le cadre logiciel fourni par " \
        "<a href=’http://www.junit.org/’>JUnit</a>: un incontournable du" \
        "d&eacute;veloppement en Java. </p>"

    for t in ${RT_TESTS_DIR}/${RT_Q}/Test_*.java
      do
         ln -sf $t ./
         rt_test_with_one_class $t $TOTAL $COMPILE_GAIN $EXECUTE_GAIN $ASSERTION_GAIN
    done
}

# Utilitaire de la fonction precedente:
#--------------------------------------
# CLASS a tester
# COMPILE_GAIN par compilation reussie d’un Test_*.java
# EXECUTE_GAIN par methode test_* reussie

function rt_test_with_one_class {
    local t=$1
    local TOTAL=$2
    local COMPILE_GAIN=$3
    local EXECUTE_GAIN=$4
    local ASSERTION_GAIN=$5

    tn=‘basename $t .java‘
    cmd="${JAVAC} $tn.java"
    rt_echo Voici le contenu du test ’<tt>’$tn’</tt>’
    rt_file_echo $tn.java
    rt_echo Je vais compiler ma classe de test ’<tt>’$tn’</tt>’ \
            avec la commande ’<tt>’$cmd’</tt>’.
    $cmd 2>${RT_TMP}/$tn.err
    if [ -s ${RT_TMP}/$tn.err ]
    then
        rt_error Il y a eu des problemes de compilation que voici:
        rt_start_stderr
        cat ${RT_TMP}/$tn.err
        rt_stop_stderr
    fi
    # OK, on a pu compiler, on execute maintenant:
    if [ -f $tn.class ]
```

```
    then
        rt_win ${COMPILE_GAIN}
        cmd="${JAVA} fr.lip6.qnc.cfskit.TestRun $tn"
        rt_echo Je vais lancer ma classe de test '<tt>'$tn'</tt>' \
                avec la commande '<tt>'$cmd'</tt>'.
        rt_start_stdout
        rt_confiner $cmd | rt_tee 50k ${RT_TMP}/$tn.out
        rt_stop_stdout
        perl -ne '
use strict;
print "$1 $2 $3 $4\n"
 if /^\[\[(+) Assertions, (+) Tests, (+) Failures, (+) Errors \]\]/;
' < ${RT_TMP}/$tn.out | { read a t f e ;
            rt_echo "&Agrave; $ASSERTION_GAIN point l'assertion et &agrave;" \
                "$EXECUTE_GAIN le test r&eacute;ussi tout entier, vous obtenez "
            rt_win "($EXECUTE_GAIN * ($t - $f - $e)) + ($ASSERTION_GAIN * $a)" ;
            if perl -e "
use strict;
my $c = ${COMPILE_GAIN}
       + ($EXECUTE_GAIN * ($t - $f - $e))
       + ($ASSERTION_GAIN * $a);
exit( $c <= $TOTAL+0.009 );
"
            then rt_internal_error "Total " \
 "${COMPILE_GAIN} + ($EXECUTE_GAIN * ($t - $f - $e)) + ($ASSERTION_GAIN * $a)" \
                 "obtenu superieur a $TOTAL."
            fi
}
    fi
}

# end of common.sh
```

et les constantes associées :

```
#! /bin/sh

# Le repertoire ou toutes les classes devraient apparaitre:
PACKAGE_DIR=fr/upmc/infop6/mlo
# L'interface que les etudiants ont a implanter
INTERFACE_NAME=Binder
# Le repertoire que les etudiants doivent soumettre via DelBul
STUDENT_DELBUL_DIR=mlo-2003nov

# Parametrage de Java

export JAVA_HOME=${JAVA_HOME:-/opt/j2sdk1.4.0}
CLASSPATH=$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar:
CLASSPATH=$JAVA_HOME/jre/lib/rt.jar:$CLASSPATH
CLASSPATH=${RT_TESTS_DIR}/junit.jar:${RT_TESTS_DIR}/cfskit.jar:${CLASSPATH}
```

```
export CLASSPATH=.:${CLASSPATH}

JAVAP="${JAVA_HOME}/bin/javap"
JAVA="$JAVA_HOME/bin/java"
JAVAC="jikes --deprecation"
#JAVAC="${JAVA_HOME}/bin/javac -deprecation"

# end of globals.sh
```

Et voici un test utilisant tout ce qui précède :

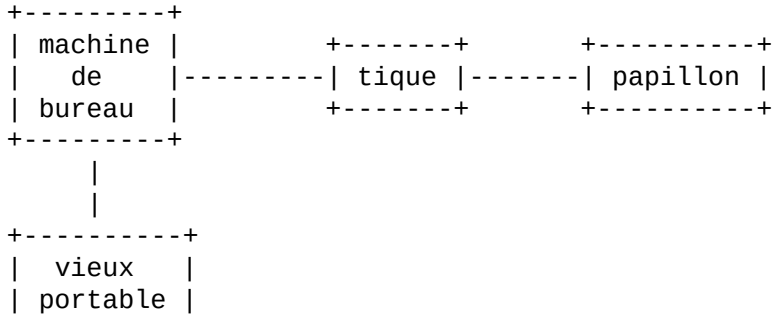
```
#! /bin/sh
# Charger la bibliotheque de fonctions communes:
. ${RT_MARK_DIR}/runtestlib.sh
# Charger les variables communes:
. ${RT_TESTS_DIR}/common.sh

rt_compile_class 1 ${PACKAGE_DIR}/Option.java

COMPILE_GAIN=0.15
EXECUTE_GAIN=0.15
ASSERTION_GAIN=0.1

rt_test_with_classes 3 ${COMPILE_GAIN} ${EXECUTE_GAIN} ${ASSERTION_GAIN}

# fin.
```

Plutôt que de réunir tous les tests en une seule classe (même comportant de multiples méthodes de test) qui, s'il manque une seule méthode dans la classe (de l'étudiant) à tester ne pourra être compilée, il vaut mieux éclater les tests en de multiples classes.

D Un exemple

Cette section raconte par le menu la correction du partiel sur machine MLO de novembre 2003. Les copies ont été ramassées par Delivery_Builder puis récupérées comme indiqué en section 4.3 ce qui mène à un fichier *arch_MLO-2003-00.4809.tgz*. La correction a été effectué sur un vieux portable déconnecté du réseau. Le reste de la procédure suit ce qui est indiquée en section 4.1.

Voici, en ASCII-art, les machines impliquées. J'édite, je mets au point sur la machine de bureau, je corrige sur le vieux portable, je diffuse sur papillon (qui n'est accessible que via tique).



```
+----------+
```

Faites attention aux invites qui indique quel utilisateur émet quelle commande. Tout ce qui suit est effectué sur la machine de notation c'est-à-dire le vieux portable. Les fichiers ont été installés sur le vieux portable par `rsync` mais ont été créés et modifiés sur une machine de bureau (correspondant à l'invite `queinnec%` ci-après).

```
root% useradd mlo2003nov
root% passwd mlo2003nov
root% mkdir ~mlo2003nov/public_html
```

Il est utile que le nom `mlo2003nov` pris soit en minuscule (pour Unix) et sans tiret (pour que ce soit un identificateur SQL valide). Ce qui suit transforme le format de l'archive en ce qui est attendu par CFSkit.

```
root% cd ~mlo2003nov/public_html
root% mkdir logs tmp
root% chown mlo2003nov.mlo2003nov logs tmp
root% mkdir students
root% tar xzf /tmp/arch_MLO-2003-00.4809.tgz
root% cd LVR_GROUPEES
root% for d in * ; do (cd $d ; tar czf ../$d.tar.gz . ) ; done
root% cd ..
root% mv LVR_GROUPEES/*.tar.gz students/
root% rm -rf LVR_GROUPEES
```

Enfin l'on établit quelques liens vers les sources du CFSkit et vers le notateur de l'épreuve. Le fichier de configuration de `make`, nommé *mlo2003nov.mkf*, va être créé sous peu ainsi que la page de présentation des résultat *index.php* qui est une copie modifiée de la page *CFSkit/src/php/index.php*. Il y a quelques variables globales en tête à mettre à jour (comme la feuille de style à utiliser, le courriel du responsable du partiel, le nom de la base et les nom et mot de passe de l'utilisateur de cette base ainsi que le préfixe des tables de données à utiliser).

```
root% ln -s /home/queinnec/Cours/Licence/CFS/src ./
root% ln -s src/php/libcfskit.php ./
root% ln -s /home/queinnec/Cours/MLO/Exam/2003Nov/site/index.php ./
root% ln -s /home/queinnec/Cours/MLO/Exam/2003Nov/tests ./
root% ln -s /home/queinnec/Cours/MLO/Exam/2003Nov/mlo2003nov.mkf ./Makefile
```

Pour résumer, voici la configuration et les droits des répertoires ainsi créés :

```
root% cd ~mlo2003nov ; ls -Rl
.:
total 4
drwxr-xr-x  4 root root 4096 avr 14 12:19 public_html/

./public_html:
total 20
lrwxrwxrwx  1 root     root   36 avr 14 12:15 cfskit -> /home/queinnec/Cours/Licence/CFS
-r--r--r--  1 queinnec spi  8837 avr 14 12:18 index.php
```

```
lrwxrwxrwx  1 root     root   24 avr 14 12:16 libcfskit.php -> cfskit/php/libcfskit.php
drwxr-xr-x  2 mlo2003nov mlo2003nov 4096 avr 14 12:12 logs/
lrwxrwxrwx  1 root     root   53 avr 14 12:18 Makefile -> /home/queinnec/Cours/MLO/Exam/
drwxr-xr-x  2 root     root 4096 avr 14 12:14 students/
lrwxrwxrwx  1 root     root   50 avr 14 12:18 tests -> /home/queinnec/Cours/MLO/Exam/200
```

Voici le premier contenu du fichier *mlo2003nov.mkf* juste pour pouvoir noter les copies des étudiants en local. Ce fichier est mis au point par le créateur de l'épreuve et donc ni `root`, ni `mlo2003nov`.

```
work :  local.mark.all.students.tgz          local.rebuild.db.on.other          dump.tab

CFSKIT_DIR      =       ${LOCAL_USER_DIR}/public_html/src

EXAMINATION     =       mlo2003nov
LOCAL_USER_DIR  =       /home/${EXAMINATION}
LOCAL_TESTS_DIR =       ${LOCAL_USER_DIR}/public_html/tests/
LOCAL_TGZDIR    =       ${LOCAL_USER_DIR}/public_html/students/
LOCAL_RUN_DIR   =       ${LOCAL_USER_DIR}/public_html/tmp/
LOCAL_LOG_DIR   =       ${LOCAL_USER_DIR}/public_html/logs/
LOCAL_SERVER_RESULTS_URL=       /~${USER}/tests/
OTHER_DB                =       dbufr1
OTHER_DBPREFIX          =       ${EXAMINATION}
OTHER_ORG_CFSKIT_DIR    =       ${CFSKIT_DIR}

include ${CFSKIT_DIR}/local.mkf
include ${CFSKIT_DIR}/run.mkf
include ${CFSKIT_DIR}/mark.mkf

# end of mlo2003nov.mkf
```

Il ne suffit plus qu'à l'utilisateur `mlo2003nov` (sur la machine de notation c'est-à-dire le vieux portable) de lancer la `make` dans le répertoire *public_html* :

```
mlo2003nov% cd ~/public_html
mlo2003nov% make
```

À partir de maintenant, il est possible de consulter la page ouèbe correspondant à l'url (encore faut-il que la base `mlo2003nov` existe et que l'utilisateur `mlo2003nov` soit créés (sur la machine de notation c'est-à-dire le vieux portable) ce qui peut toujours être effectué avec l'utilisateur `postgres`).

```
postgresql% createuser -A -D mlo2003nov
postgresql% createdb mlo2003nov -Elatin1
```

Il est alors temps de consulter les résultats de la notation avec l'URL suivante qui extirpe les données de la base (locale à la machine de notation) :

```
http://la.machine/~mlo2003nov/
```

Après quelques réglages (installer `jikes`, régler le `CLASSPATH`, etc.) on arrive à des notes non systématiquement égales à zéro. Si l'on ajoute des tests, il est bon de noter d'abord les pseudo-copies ce qui est rapide plutôt que de relancer la notation complète (ce qui prend une dizaine de minutes) pour éliminer les petites bévues. Cette vérification s'effectue comme suit (vérifier que la copie parfaite obtient bien toujours le maximum des points) :

```
queinnec% make pseudos.mark
queinnec% konqueror /tmp/mlo2003nov/logs/index.html
queinnec% rsync -avu . vieuxportable:     # A adapter!
```

Quand on commence à avoir des notes montrables, on peut avoir envie de les partager avec les autres membres de l'équipe pédagogique. Quelques lignes de plus dans un *Makefile* suffisent. Voici celles que j'ai ajouté (faute de mieux) à *mlo2003nov.mkf* :

```
# ############## Pour transferer les resultats sur tique:
# D'abord on rapatrie sur la machine de bureau et ensuite on transmet
# sur tique (via une machine du lip6 (pour sauvegarde)).
# Pour tourner cette commande:
#         make -f mlo2003nov.mkf CFSKIT_DIR=cfskit install.on.site

RSYNC   =       rsync --rsh ssh
RSH     =       ssh -X -A -g -t

NOTE_SERVER     =       porbou.vld7net.fr
MIDDLE          =       youpou.lip6.fr
MIDDLEPATH      =       Cours/MLO/Exam/2003Nov
REMOTE          =       tique.infop6.jussieu.fr

LOCAL_SERVER            =       ${REMOTE}
LOCAL_SERVERPATH        =       /Ens/maitrise/2003/algoprog/mlo/public
LOCAL_SERVERDB  =       dbufr1 -h papillon.infop6.jussieu.fr

install.on.site :
        @echo '+++++++++++++++++' rapatrier les resultats de correction ...
        ${RSYNC} -avu --exclude tmp \
                --exclude '*.log' --exclude '*.sql' --exclude '*.toc' \
                ${NOTE_SERVER}:${LOCAL_USER_DIR}/public_html ./
        ${RSYNC} -avu ${NOTE_SERVER}:/tmp/${EXAMINATION}.sql public_html/
        @echo '+++++++++++++++++' renommer en ${EXAMINATION} ...
        -rm -rf ${EXAMINATION}
        mv public_html ${EXAMINATION}
        @echo '+++++++++++++++++' transmettre sur youpou ...
        ${RSYNC} -avuz --delete ${EXAMINATION} ${MIDDLE}:${MIDDLEPATH}/
        @echo puis transmettre sur tique ...
        ${RSH} ${MIDDLE} "cd ${MIDDLEPATH}/ && \
        ${RSYNC} -avu -cLR --delete ${EXAMINATION} \
                ${LOCAL_SERVER}:${LOCAL_SERVERPATH}/"
        @echo '+++++++++++++++++' regenerer les tables dans la base du serveur
        ${RSH} ${MIDDLE} ${RSH} ${LOCAL_SERVER} psql ${LOCAL_SERVERDB} \
        -f ${LOCAL_SERVERPATH}/${EXAMINATION}/${EXAMINATION}.sql
        @echo '+++++++++++++++++' Pret sur ${REMOTE}.
```

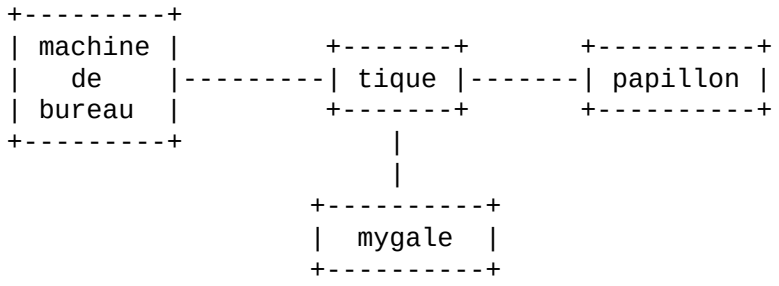
porbou est mon vieux portable, youpou ma machine au LIP6, tique est la machine des enseignants de l'UFR d'informatique qui permet d'accéder aux partitions servies par le démon HTTPD sur le serveur ouèbe pédagogique papillon. Maintenant les notes et les copies sont consultables avec l'URL :

`http://www.infop6.jussieu.fr/maitrise/2003/algoprog/mlo/public/mlo2003nov/`

E Un autre exemple

Cette section raconte par le menu la correction des copies du cfs du 17 décembre 2003 matin. Les copies ont été ramassées par Delivery_Builder puis récupérées comme indiqué en section 4.3 ce qui mène à un fichier *arch_cct1-cfs-2003dec-00.1859.tgz*. La correction a été effectué sur la machine de confinement mygale car les copies sont certainement plus « sensibles ».

Voici, en ASCII-art, les machines impliquées. J'édite, je mets au point sur la machine de bureau, je corrige sur la machine de confinement mygale, je diffuse sur papillon ; ces deux dernières ne sont accessibles que via tique.



Sur ma machine de bureau, les fichiers liés au CFS sont dans un répertoire *CFS2003dec17/*. J'enrichis donc le *Makefile* qui s'y trouve pour qu'à côté de la correction des pseudo-copies et de l'installation des énoncés et fichiers adventices du CFS sur papillon, s'ajoutent les règles pour la correction sur la machine de confinement.

Mais pour pouvoir correctement paramétrer les différents fichiers du CFSkit, il faut, depuis tique, créer et remplir quelques répertoires visibles depuis mygale. Il faut essentiellement trois choses pour corriger sur la machine de confinement :

1. les copies (ici obtenues par Delivery_Builder puis recopiées dans */tmp/* et adaptées au format souhaité par CFSkit)
2. les sources du CFSkit(déjà présents)
3. les tests (recopiés dans la hiérarchie de répertoires du site de licence car ils seront consultables par les étudiants). La partie du site de licence consacrée au CFS est en */Ens/licence/2003/cct/CFS2003dec17am/* et donc accessibles à partir de l'URL http://www.infop6.jussieu.fr/licence/2003/cct/CFS2003dec17am/.

```
tique% mkdir /Infos/licence/2003/cct1/2003dec17am
tique% cd /Infos/licence/2003/cct1/2003dec17am
tique% tar xzf /tmp/arch_cct1-cfs-2003dec-00.1859.tgz
tique% mkdir students
tique% cd LVR_GROUPEES
tique% for d in * ; do (cd $d ; tar czf ../$d.tar.gz . ) ; done
tique% cd ..
tique% mv LVR_GROUPEES/*.tar.gz students/
```

```
tique% rm -rf LVR_GROUPEES
tique% chmod -R a+r .
tique% tar xzf /Ens/licence/auth/private/delbul/cfskit.tgz
tique% rm -rf cfskit-2003Dec15/TME
tique% cp -rp cfskit-2003Dec15/src ./
tique% cp -rp /Ens/licence/2003/cct/CFS2003dec17am/tests ./
```

Voici la configuration du *Makefile* :

```
SERVER_RESULTS_URL=/licence/2003/cct1/${EXAMINATION}
OTHER_TRANSIT_DIR=/Infos/licence/2003/cct1/${EXAMINATION}
OTHER_ORG_TGZDIR=${OTHER_TRANSIT_DIR}/students
OTHER_RESULTS_DIR=/Ens${SERVER_RESULTS_URL}
OTHER_ORG_CFSKIT_DIR=${OTHER_TRANSIT_DIR}/cfskit-2003Dec15/src
OTHER_ORG_TESTS_DIR=${OTHER_RESULTS_DIR}/tests
OTHER_DB=dbufr1 -h papillon
OTHER_DBPREFIX=${EXAMINATION}
CONFINER_CFSKIT_DIR=${OTHER_TRANSIT_DIR}/src
CONFINER_TGZDIR=${OTHER_ORG_TGZDIR}
CONFINER_TESTS_DIR=${OTHER_TRANSIT_DIR}/tests
CONFINER=mygale.infop6.jussieu.fr
CONFINER_USER=test91
CONFINER_MAKEFILE=${OTHER_TRANSIT_DIR}/Makefile

include ${CFSKIT_DIR}/mark.mkf

install.makefile :
        cd && ${RSYNC} -avu ${DIR}/Makefile ${MIDDLE}:${DIR}/
        ${RSH} ${MIDDLE} \
                "${RSYNC} -avu ${DIR}/Makefile ${RIGHT}:${OTHER_TRANSIT_DIR}/"

install.and.initialize : install.makefile
        ${RSH} ${MIDDLE} "${RSH} ${RIGHT} 'cd ${OTHER_TRANSIT_DIR}/ && \
        make CFSKIT_DIR=src before.marking.on.other' "

install.and.mark : install.makefile
        ${RSH} ${MIDDLE} "${RSH} ${RIGHT} 'cd ${OTHER_TRANSIT_DIR}/ && \
        time make CFSKIT_DIR=src re.marking.on.other after.marking.on.other' "
```

Pour enchaîner la notation sur mygale sans avoir à redonner à tout bout de champ le mot de passe de l'utilisateur, il suffit de recopier sa clé publique de tique vers mygale.

```
tique% scp -rp ~/.ssh test91@mygale:
test91@mygale's password:
authorized_keys      100% |*****************************|   349       00:00
authorized_keys2     100% |*****************************|   858       00:00
known_hosts          100% |*****************************|  8725       00:00
known_hosts2         100% |*****************************|  2473       00:00
random_seed          100% |*****************************|   512       00:00
```

Enfin on peut lancer la notation. Depuis sa machine de bureau avec :

```
bureau% make install.and.mark
```

ou depuis la machine de confinement même (regardez aussi les règles présentes dans *mark.mkf* comme `nohup.mark.all.studen` ou `re.mark.all.students.tgz`):

```
mygale% make -f /Infos/licence/2003/cct1/CFS2003dec17am/Makefile \
   CFSKIT_DIR=/Infos/licence/2003/cct1/CFS2003dec17am/src \
   mark.all.students.tgz
```

Quand tout est corrigé, les résultats peuvent être exportés sur le serveur ouèbe avec :

```
tique% cd /Infos/licence/2003/cct1/2003dec17am
tique% make CFSKIT_DIR=src after.marking.on.other
```

L'URL http://www.infop6.jussieu.fr/licence/2003/cct/CFS2003dec17am/ permet alors de consulter les notes et les copies.