Licence d'informatique
Module de C/Unix

Projet Unix

comp — Check out my programs

Philippe MARQUET

Octobre 2002

L'objet de la commande `comp` (Check out my programs) que vous allez développer est de valider un programme par une étude de son exécution. C'est-à-dire vérifier que pour une exécution donnée d'un programme, le résultat produit est conforme au résultat attendu.

Systématiser de telles vérifications lors d'une phase de développement d'un logiciel permet de
- réaliser effectivement des tests. Combien de logiciels n'ont que trop succinctement été testés !
- focaliser sur les tests devant être menées plutôt que sur la manière de les mener ;
- rejouer systématiquement une batterie de tests lors de la modification du logiciel.

Une première partie du sujet présente une version basique de la commande `comp`. Des extensions sont ensuite proposées.

1 Une exécution

L'exécution d'une commande est définie par la commande elle-même, les entrées de la commande et les sorties de la commande.

- La commande est identifiée par le nom d'un fichier exécutable.
- Les entrées de la commande sont composées
  1. des paramètres passées sur la ligne de commande ;
  2. de données lues par la commande sur son entrée standard `stdin`.
- Les sorties de la commandes sont composées
  1. des données produites par la commande sur sa sortie standard `stdout` et sa sortie d'erreur `stderr` ;
  2. du statut retourné par l'exécution de la commande.

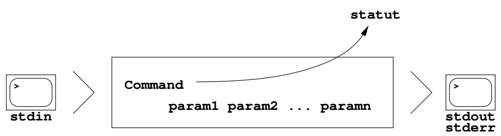

FIG. 1 – Entrées et sorties d'une exécution

La figure 1 schématise les entrées et les sorties d'une exécution.

2 Exécution valide

Une exécution est valide si et seulement si pour des entrées données elle produit les sorties attendues. C'est-à-dire si et seulement si
- pour des paramètres donné sur la ligne de commande ;
- pour des valeurs données lues sur l'entrée standard ;

l'exécution produit
- des valeurs attendues sur la sortie standard et la sortie d'erreur ;

et se termine avec un statut attendu.

Id: comp.tex,v 1.2 2002/10/04 05:48:08 marquet Exp

3 Configuration d'une exécution

Une exécution va ainsi pouvoir être identifiée par un certain nombre de paramètres :

`BINARY` le nom de l'exécutable ;

`PARAMS` les paramètres de la ligne de commande

`STDIN` (ou `STDINFILE`) la valeur (ou[1] le nom du fichier qui contient cette valeur) qui sera lue sur l'entrée standard lors de l'exécution ;

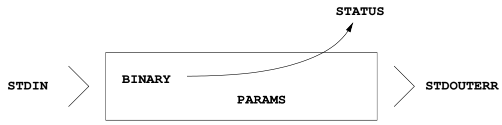

FIG. 2 – Configuration d'une exécution

`STDOUTERR` (ou `STDOUTERRFILE`) la valeur (ou le nom du fichier contenant cette valeur) qui devrait être produite par la commande sur la sortie standard et la sortie d'erreur ;

`STATUS` la valeur du statut qui devrait être retournée par l'exécution ;

D'autres paramètres sont aussi disponibles :

`DIRSTD` un répertoire à partir duquel les noms des fichiers `STDINFILE` et `STDOUTERRFILE` sont relatifs ;

`SIGNIFICANTSPACES` une valeur booléenne (`YES` ou `NO`) qui indique si lors de la comparaison des résultats produits et des résultats attendus les caractères espaces sont significatifs.

4 Fichier de configuration

Notre commande `comp` est paramétrée par un fichier de configuration. Un fichier de configuration est constitué d'une suite de sections.

Il existe une section générale et une section par exécution à valider. La section générale est placée en début du fichier de configuration. Le début de la section relative à une exécution de nom `test` est marquée par une ligne

```
[test]
```

Une section est formée de lignes de la forme

```
parameter = value
```

Les lignes vides (i.e. ne comportant que des espaces) et les lignes dont le premier caractère significatif est un dièse # sont ignorées.

Un fichier de configuration produit deux informations :

1. Une liste de tests pouvant être validés.
2. La valeur d'un certain nombre de paramètres. Pour un test, la valeur d'un paramètre donné est définie dans la section de l'exécution ; à défaut dans la section principale.

   Certains paramètres exclusifs se masquent l'un l'autre ; par exemple si la section principale définie `STDIN` à une certaine valeur, la définition de `STDINFILE` dans la section d'un test donné masque cette première définition.

Exemple de fichier de configuration :

```
# Fichier de configuration pour un premier test de ma commande comp

# la commande a tester par defaut
BINARY = cat

# Configuration generale
SIGNIFICANTSPACES = YES
```

[1] Il s'agit bien entendu d'un ou exclusif.

```
# un test elementaire qui doit etre valide
[arith]
BINARY = expr
PARAMS = 2 + 3
STDOUTERR = 5

# un autre test elementaire qui doit etre valide
[print]
PARAMS = /etc/passwd
STDOUTERRFILE = /etc/passwd

# une variante du precedent
[print2]
PARAMS = /etc/passwd
DIRSTD = /etc
STDOUTERRFILE = passwd

# un test qui sera invalide
[false-is-false]
BINARY = false
STATUS = 0

# un  test qui doit etre invalide
[group-is-nil]
PARAMS = /etc/passwd /etc/group
STDOUTERRFILE = /etc/passwd
```

5 Syntaxe et options de la commande **`comp`**

La syntaxe de la commande `comp` est la suivante :

```
comp [options]... [test-name]...
```

on peut ainsi valider un ou plusieurs tests en une invocation de la commande. Les tests dont le nom est donné sur la ligne de commande sont traités. Par défaut, l'ensemble des tests du fichier de configuration est traité.

Les options sont les suivantes :

`-c cfg-file` considère le fichier de configuration *`cfg-file`*. La valeur par défaut est `comp.cfg`.

`-d dir` recherche le fichier de configuration dans le répertoire *`dir`*; par défaut dans le répertoire courant.

`-v` affiche sur la sortie standard la version de la commande `comp` et termine.

`-s` | **`-m`** | **`-l`** format court, moyen, ou long (*short*, *medium*, *long*). Par défaut `-m`. Le format court n'affiche que la liste des tests non validés. Le format moyen affiche pour chacun des tests s'il a été validé ou non. Le format long affiche en plus pour les tests non validés la différence entre le résultat attendu et le résultat effectif.

La commande `comp` se termine sur un succès si et seulement si tous les tests traités ont été validés.

6 Extensions de la commande **`comp`**

Les sections précédentes présentent une version minimale de la commande `comp`. Nous définissons ici les extensions que vous pourrez intégrer dans une seconde version de vos développements. Dans un premier temps, ne vous arrêtez pas sur ces extensions.

Distinguer **stdout** et **stderr**

On veut pouvoir distinguer des données produites par la commande sur la sortie standard et la sortie d'erreur. Pour ce faire on définit quatre nouveaux paramètres : STDOUT, STDOUTFILE, STDERR, et STDERRFILE identifiant le contenu attendu sur la sortie standard et sur la sortie d'erreur. De plus un paramètre booléen MIXEDOUTERR indique si on doit considérer la sortie standard et la sortie d'erreur indépendamment.

Lecture et écriture depuis la commande

La commande à tester peut produire des entrées/sorties directement dans des fichiers autres que ses sorties standard.

La valeur NINFILE et la valeur NOUTFILE indiquent respectivement combien de fichiers sont lus et écrits par la commande.

Une série de paramètres INFILENAME<*i*> et INFILEDATA<*i*> identifie les fichiers lus par la commande. La commande va lire chacun des fichiers de nom INFILENAME<*i*> dont le contenu doit être égal au contenu du fichier INFILEDATA<*i*> avant l'exécution pour le test.

Une série de paramètres OUTFILENAME<*i*> et OUTFILEDATA<*i*> identifie les fichiers écrits par la commande. La commande va écrire chacun des fichiers de nom OUTFILENAME<*i*> dont le contenu attendu après l'exécution doit être égal au contenu du fichier OUTFILEDATA<*i*>.

La valeur du paramètre DIRFILE est un répertoire à partir duquel les noms de fichier INFILENAME<*i*> et OUTFILENAME<*i*> sont relatifs. La valeur du paramètre DIRDATA est un répertoire à partir duquel les noms de fichier INFILEDATA<*i*> et OUTFILEDATA<*i*> sont relatifs.

Compression

Un paramètre booléen COMPRESSEDDATA identifie le fait que les fichiers STD*FILE et *FILEDATA<*i*> sont compressés. La commande de décompression à utiliser qui lit son entrée standard et produit un résultat sur sa sortie standard est zcat. Le paramètre ZCAT peut surcharger cette valeur par défaut.

Permutation

Il s'agit de proposer une extension de la comparaison des contenus attendu et produit. Le paramètre booléen UNSORT indique que ces contenus doivent être identiques à une permutation des lignes près.

Variables

On considère que les valeurs données aux paramètres doivent être évaluées par un shell. Ainsi il est possible d'utiliser des variables du shell dans un fichier de configuration par exemple :

```
DIRSTD = $HOME/tests
```

7 Votre projet

Développements

Dans un premier temps, il s'agit de se construire des outils, par exemple sous forme de fonctions Bourne shell : identification du fichier de configuration, identification de l'ensemble des tests d'un fichier de configuration donné, identification des paramètres pour un test donné...

Ensuite, vous implanterez la version de base de la commande `comp`. Enfin vous pourrez étendre progressivement cette version pour intégrer les fonctionnalités supplémentaires. À chacune de ces étapes, il sera nécessaire de valider votre commande par des essais bien choisis.

Rendre votre projet

Lors d'une démonstration de votre logiciel, vous devrez rendre :

- le source Bourne shell de votre commande `comp` ;
- un jeu de fichiers de configuration, tests de votre commande ;
- un brève documentation de votre commande. Cette documentation comporte :
  - une description des fonctionnalités de votre commande destinée à un utilisateur,
  - une description de votre implantation de la commande `comp`. Elle est destinée à un développeur qui désirerait étendre ou corriger votre implantation.

Cette documentation est à rendre sous forme de fichiers textes ou de fichiers HTML *simples*.