1

Rédiger et lancer un premier script

L'objectif de cette leçon est de se familiariser rapidement avec Lua. Si vous avez déjà programmé dans un autre langage, vous retrouverez certainement des éléments de syntaxe familiers.

1. Hello World

Dans l'apprentissage d'un nouveau langage de programmation, la tradition veut que le tout premier programme écrit doive afficher « Hello World! ». Nous ne dérogerons pas à cette longue tradition. En Lua, c'est trivial :

```
print("Hello World!")
```

Comment lancer cet exemple ? Il y a au moins deux possibilités, dans une console interactive ou avec un fichier extérieur.

Interpréteur interactif

Commençons donc par lancer l'interpréteur Lua. Sous Unix, cela se fait simplement avec la commande `lua` depuis le *shell*. Sous Windows, suivant la distribution que vous avez installée, il est normalement possible de faire de même depuis une fenêtre exécutant `cmd.exe`, mais il devrait également exister un raccourci depuis le menu principal.

Dans tous les cas, vous devriez voir une notice de version et une invite de ligne de commande telles que :

```
Lua 5.2.0  Copyright (C) 1994-2011 Lua.org, PUC-Rio
>
```

Si la version affichée est supérieure à 5.2.0, aucun problème, vous avez juste une version de Lua plus récente que cet ouvrage. Si c'est 5.1.x, soyez attentif au fait que dans la rédaction nous avons privilégié la version 5.2 de Lua, mais que là où il y a des diffé-

rences, des notes séparées expliquent comment faire avec la version précédente 5.1. Si vous utilisez encore Lua 5.0 ou antérieur, il est grand temps d'effectuer une mise à jour !

***Note** › Si le message initial ressemble plutôt à cela :*

```
LuaJIT 2.0.0-beta10 -- Copyright (C) 2005-2012 Mike Pall. http://
luajit.org/
JIT: ON CMOV SSE2 SSE3 SSE4.1 fold cse dce fwd dse narrow loop abc
 fuse
>
```

c'est que vous avez installé LuaJIT et non Lua (l'original). LuaJIT est une réécriture complète de l'interpréteur par Mike Pall, qui parvient à générer du code machine à la volée dans les boucles d'exécution critiques. LuaJIT est en principe compatible avec Lua (syntaxe, librairies ainsi que l'API), mais au moment de la rédaction de ces lignes, la version 2.0.0-beta10 n'est pas encore compatible 5.2, bien qu'elle intègre déjà quelques-unes des nouveautés. Référez-vous donc à la documentation officielle.

Dans l'invite de commande, vous pouvez maintenant entrer `print("Hello World!")` suivi d'un retour à la ligne. Le message s'affiche et un nouveau caractère `>` indique que l'interpréteur est prêt à recevoir une nouvelle ligne de commande. Sur certains systèmes d'exploitation, les flèches du clavier permettent de récupérer les lignes de commandes déjà introduites, afin de les modifier facilement.

***Note** › Si vous avez l'habitude d'utiliser la touche TAB pour compléter les mots commencés, vous serez déçu de constater que cela ne fonctionne (normalement) pas dans l'interpréteur Lua. Si cela vous dérange, sachez qu'au moins un* patch *écrit par Mike Pall ainsi qu'un librairie plus complète nommée luarlcompleter permettent l'utilisation de* Readline *comme moteur de complétion.*

Dans l'interpréteur interactif, si vous entrez `return "Hello World!"`, ou plus simplement encore `="Hello World!"` vous obtiendrez le même résultat. Le signe `=` en début de ligne est remplacé par `return` au moment de la saisie ; ce n'est pas une expression Lua valide. Dans un script Lua par exemple, un signe égal au début d'une instruction génère une erreur de syntaxe. Néanmoins, c'est un raccourci fort pratique, en particulier comme calculatrice : essayez par exemple `=10+3*2^4`. En revanche, contrairement à beaucoup de consoles interactives d'autres langages, le fait d'entrer une simple expression telle que `2+3` n'affichera pas `5` mais une erreur de syntaxe. Prenez donc l'habitude de faire précéder ce genre d'expression par `=`.

***Astuce** › Pour quitter l'interpréteur, selon votre système d'exploitation, tapez `Ctrl+C`, `Ctrl+D` ou `Ctrl+Z`. Vous pouvez également entrer la commande `os.exit()`. Sous Windows, il est également possible de simplement fermer la fenêtre à l'aide de la souris...*

Script sur disque

Écrire du code directement dans l'interpréteur interactif n'est utile que pour des petits essais rapides. Dès lors que vous souhaitez réellement programmer, il devient indispensable d'enregistrer le code Lua dans un ou plusieurs fichiers.

Pour cela, ouvrez votre éditeur de texte favori. Si vous n'avez jamais travaillé avec ce genre de logiciel, référez-vous au chapitre d'installation pour des conseils. Certains éditeurs de texte intègrent par défaut une aide à la programmation en Lua, sous forme de coloration syntaxique, de liste des fonctions ou encore de complétion des mots clés. Sur d'autres (par exemple *UltraEdit* ou *Visual Studio*), il vous suffit de télécharger des greffons pour supporter ce langage.

Dans l'éditeur de texte, ouvrez un nouveau document, saisissez `print("Hello World!")` puis enregistrez le fichier `hello.lua` où bon vous semble. L'extension `.lua` n'est pas obligatoire, mais il est d'usage de l'utiliser pour les scripts en Lua.

Pour lancer le programme, ouvrez un terminal (typiquement *bash* ou *cmd*) et entrez simplement :

```
lua chemin/vers/script/hello.lua
```

Évidemment, il est souvent plus simple de commencer par changer de répertoire courant (avec `cd chemin/vers/script`), pour ensuite ne taper que `lua hello.lua`.

Sous Unix, il est possible d'utiliser le mécanisme de *shebang*, c'est-à-dire de commencer le fichier par `#!` suivi du chemin vers l'interpréteur Lua. Dans ce cas, n'oubliez pas d'enclencher l'attribut d'exécution pour le fichier. Exemple :

```
$ cat > hello.lua
#!/usr/local/bin/lua
print("Hello World!")
^D
$ chmod +x hello.lua
$ ./hello.lua
Hello World!
```

Sous Windows, la plupart des programmes d'installation associent l'extension `.lua` avec l'interpréteur Lua. Il suffit donc de double-cliquer sur un fichier `.lua` pour lancer le script. Il y a toutefois un problème récurrent avec cette méthode. Généralement, la fenêtre d'exécution se ferme avant que vous n'ayez le temps de lire le résultat à l'écran !

Astuce *› Pour éviter cette fermeture prématurée, vous pourriez terminer chaque script avec l'instruction* `io.read()`*, qui va attendre un retour à la ligne en fin d'exécution. Toutefois ce procédé n'est pas recommandé, car il gêne le lancement du programme depuis la ligne de commande. Il vaut mieux modifier la ligne de commande associée à* `.lua` *dans la base de registre de Windows. Voici comment faire depuis une fenêtre* cmd *lancée en tant qu'administrateur :*

```
assoc .lua=Lua.Run
ftype Lua.Run=cmd /k ""C:\Program Files\Lua\5.2\lua.exe" -i "%1""
```

Bien entendu, vérifiez le répertoire où se trouve Lua sur votre machine. L'option `-i` *de l'interpréteur Lua demande d'entrer en mode interactif à la fin du script. Cela peut être utile pour consulter des variables globales après l'exécution. Et le lancement de* `cmd.exe` *est nécessaire pour que la fenêtre ne se ferme pas non plus en cas d'erreur.*

Vous pouvez également exécuter le script depuis l'interpréteur interactif, en utilisant la fonction standard `dofile`, comme ceci :

```
dofile("chemin/vers/script/hello.lua")
```

Astuce *› Sous Windows, évitez d'utiliser la barre oblique inversée* `\` *comme séparateur de répertoire, préférez la barre oblique normale* `/` *. En effet, le noyau de Windows traite indifféremment ces deux caractères. Or, d'une part, l'usage de* `\` *pose des problèmes de compatibilité avec Unix, d'autre part, le caractère* `\` *sert, comme en C, de caractère d'échappement dans les chaînes, et doit être doublé. Vous devriez donc entrer* `dofile("chemin\\vers\\script\\hello.lua")`*, ce qui est inutilement lourd. Une alternative est d'utiliser la syntaxe pour les chaînes de caractères longues :* `dofile([[chemin\vers\script\hello.lua]])`*.*

Note *› La fonction standard* `dofile` *n'accepte étonnamment qu'un seul argument, le nom du fichier. Si vous souhaitez exécuter un script en lui passant des arguments supplémentaires, il faudra recourir à la fonction* `loadfile` *sous la forme suivante :*

```
assert(loadfile("chemin/vers/script/hello.lua"))(arg1,arg2,...)
```

2. Variables

Les variables jouent un rôle essentiel dans la plupart des langages de programmation, Lua ne faisant pas exception à cette règle. Il est important de comprendre que Lua est un langage *dynamiquement typé*, ce qui signifie qu'une variable n'a pas de type associé. En revanche, sa *valeur* possède un type. Dans l'exemple qui suit, voyons comment référencer des variables :

```
-- Un commentaire simple commence par deux tirets.
-- et court jusqu'à la fin de la ligne.
```