tutoriel

Comment déboguer un script bash ?

Introduction

Tout programme doit être exempt de bogues avant d'atteindre le consommateur.
Les développeurs de logiciels font de leur mieux pour créer des programmes exempts de bogues.
Mais il est difficile de rendre le code parfait lorsqu'il y a des milliers de lignes.
Le débogage est un processus continu ; il vous aide à repérer immédiatement les bogues, à recueillir des informations précieuses sur le code et à éliminer les extraits de code redondants.
Tous les langages de programmation ont des approches communes et différentes pour trouver des bogues.
Par exemple, les programmes de débogage peuvent être utilisés pour corriger rapidement les bogues.
Alors que les scripts shell n'ont pas d'outil spécial pour déboguer le code.
Cet article traite de diverses techniques de débogage qui peuvent être utilisées pour créer un script bash sans erreur.
Avant de plonger dans les méthodes, comprenons les wrappers et comment les écrire :

Qu'est-ce qu'un shell sous Linux ?

Lorsque vous démarrez l'ordinateur, le noyau obtient des informations sur le matériel connecté et permet aux autres composants connectés de communiquer.
De plus, il gère la mémoire, le processeur, et reconnaît tout nouveau périphérique.
En général, le noyau est l'épine dorsale de tout système d'exploitation.
Mais avez-vous déjà pensé à interagir directement avec le noyau, en lui donnant une commande pour effectuer une tâche spécifique ?
Est-ce même possible ?
Tout à fait !
Avec le shell, un programme informatique doté d'une interface interactive, n'importe qui peut contrôler le noyau.
Le shell permet aux utilisateurs d'interagir avec le noyau et de lui demander d'effectuer n'importe quelle tâche.

Il existe deux shells principaux sous Unix, le **shell Bourne** et le **shell C**.
Ces deux types ont leurs propres sous-catégories.
Les différents types de shells Bourne sont **Korn Shell (ksh), Almqvist Shell (Ash), Bourne Shell Again (bash)** et **Z Shell (zsh)**.
Dans le même temps, le shell C a ses propres sous-catégories comme **C Shell (csh)** et **TENEX C Shell (tcsh)**.
Comme mentionné ci-dessus, de tous les shells, **Bash (Bourne shell à nouveau)** est le shell le plus largement utilisé et est livré en standard sur de nombreuses distributions Linux en raison de son efficacité et de sa convivialité.

Bash est le shell par défaut de nombreuses distributions Linux et est largement utilisé par des millions d'utilisateurs Linux.
Il est si varié et puissant qu'il peut effectuer n'importe quelle tâche que vous effectueriez

normalement dans des applications basées sur une interface graphique.
Vous pouvez modifier des fichiers, gérer des fichiers, afficher des photos, écouter de la musique, lire des vidéos et plus encore.

Qu'est-ce qu'un script shell ?

Puisque nous avons vu l'idée de base d'un shell, passons maintenant à l'écriture de scripts shell.
Un script shell est un programme informatique qui exécute plusieurs commandes dans un shell qui agit comme un interprète pour exécuter une fonction spécifique.
Comme mentionné ci-dessus, il existe 2 types spécifiques de shells.
Cependant, ce guide se concentre sur le shell Bourne Again (Bash).
Alors, qu'est-ce qu'un script bash ?
Sous Linux, toutes les commandes bash sont stockées dans les dossiers **/usr/bin** et **/bin**.
Par exemple, chaque fois que vous exécutez une commande, bash vérifie si elle existe ou non dans le répertoire.
La commande s'exécutera si elle trouve dans les répertoires, sinon elle générera une erreur.

Que diriez-vous de faire une tâche qui nécessite l'exécution de plusieurs commandes dans le terminal ?
Dans cette situation particulière, les scripts bash peuvent vous aider.
Le script bash est une forme de script shell qui vous permet d'avoir des programmes qui exécutent plusieurs commandes bash pour effectuer une tâche spécifique.

Quelles sont les erreurs dans les scripts bash ?

Lorsque vous travaillez avec des scripts bash ou tout autre langage de programmation, vous rencontrez de nombreux bogues.
Un bogue est un bogue ou un problème dans un programme qui peut entraîner une exécution incorrecte du programme.

Chaque langage de programmation a sa propre routine de vérification des bogues ; de même, bash possède également de nombreuses options intégrées pour déboguer un programme de terminal.

La gestion des erreurs et le débogage du programme sont tout aussi fastidieux.
C'est un travail qui prend du temps et qui peut s'aggraver si vous ne savez pas quels sont les bons outils pour déboguer votre programme.
Cet article est un guide complet sur le débogage des scripts bash pour rendre votre script sans erreur. Alors, commençons.

Comment déboguer un script bash

Lorsque vous travaillez sur de grands projets de programmation, vous rencontrez de nombreux bugs ou problèmes.
Le débogage d'un programme peut parfois être délicat.

Les programmeurs utilisent généralement des outils de débogage, et de nombreux éditeurs de code aident également à trouver des bogues en mettant en évidence la syntaxe.

Linux dispose de divers outils pour déboguer le code, tels que GNU Debugger alias gdb.
Des outils comme GDB sont utiles pour les langages de programmation qui se compilent en binaires.
Puisque bash est un langage interprété simple, il n'y a pas besoin d'outils lourds pour le déboguer.

Il existe différentes méthodes traditionnelles pour déboguer le code de script bash et l'une d'entre elles consiste à ajouter une **assertion**.
Les assertions sont des conditions qui sont ajoutées aux programmes pour tester certaines conditions et exécuter le programme en conséquence.
Il s'agit d'une méthode défensive qui aide également à trouver des bogues et à effectuer des tests.
De nombreux outils permettent d'ajouter des assertions aux scripts bash.

Eh bien, l'ajout d'assertions est l'une des vieilles astuces traditionnelles.
Des ensembles de drapeaux/options sont disponibles dans bash pour déboguer un script bash.
Ces options peuvent être ajoutées au le shebang dans le script, ou ajoutées lors de l'exécution dans le terminal.

Voyons donc les différentes méthodes de bash pour déboguer un script bash.

Comment déboguer un script bash avec l'option verbose (-v)

L'une des approches les plus simples pour déboguer un script bash consiste à utiliser l'option -v, également connue sous le nom de verbose.

L'option peut être ajoutée à un shebang ou explicitement spécifiée avec le nom de fichier du script lors de son exécution.

L'option verbose exécutera et imprimera chaque ligne de code en tant que processus d'interprétation.

Expliquons cela avec un exemple de script bash :

b_script.sh

```
#! /bin/bash
echo "Enter number1"
read number1
echo "Enter number2"
read number2
if [ "$number1" -gt "$number2" ]
then
echo "Number1 greater than number2"
elif [ "$number1" -eq "$number2" ]
then
echo "Number1 is equal to Number2"
else
echo "Number2 is greater than Number1"
fi
```

Le code ci-dessus reçoit deux nombres de l'utilisateur, puis exécute des instructions conditionnelles pour vérifier si le nombre est supérieur, inférieur ou égal à un autre nombre saisi.

Bien que n'importe quel éditeur de texte puisse être utilisé pour écrire des scripts bash, j'utilise l'éditeur Vim.

Vim est un éditeur puissant et riche en fonctionnalités qui met l'accent sur la syntaxe des scripts bash et réduit les risques d'erreurs de syntaxe.

Si vous n'avez pas l'éditeur Vim, **installez vim**

```
...@...:~$ sudo apt install vim
```

Créez un fichier de script bash en utilisant :

```
...@...:~ $ vim b_script.sh
```

Si vous êtes nouveau dans l'éditeur Vim, je vous recommande d'apprendre à utiliser l'éditeur vim avant de continuer.

Revenons maintenant au script, exécutez le script en utilisant l'option **-v** :

```
...@...:~ $ bash -v /media/tmp/b_script.sh
#! /bin/bash
echo "Enter number1"
Enter number1
read number1
3
echo "Enter number2"
Enter number2
read number2
5
if [ "$number1" -gt "$number2" ]
then
echo "Number1 greater than number2"
elif [ "$number1" -eq "$number2" ]
then
echo "Number1 is equal to Number2"
else
echo "Number2 is greater than Number1"
fi
Number2 is greater than Number1
```

Sur la sortie ci-dessus, vous pouvez voir que chaque ligne du script est imprimée sur le terminal au fur et à mesure qu'elle est traitée par l'interpréteur.

Notez que le script s’arrêtera pour accepter les entrées de l'utilisateur, puis traitera la ligne suivante du script.

Comme indiqué ci-dessus, l'option -v peut être placée après le shebang, comme indiqué ci-dessous :

b_script.sh

```
#! /bin/bash -v
echo "Enter number1"
read number1
echo "Enter number2"
read number2
if [ "$number1" -gt "$number2" ]
then
echo "Number1 greater than number2"
elif [ "$number1" -eq "$number2" ]
then
echo "Number1 is equal to Number2"
else
echo "Number2 is greater than Number1"
fi
```

De même, un drapeau verbose peut également être ajouté à la prochaine ligne shebang à l'aide de la commande **set** :

b_script.sh

```
#! /bin/bash
set -v
echo "Enter number1"
read number1
echo "Enter number2"
read number2
if [ "$number1" -gt "$number2" ]
then
echo "Number1 greater than number2"
elif [ "$number1" -eq "$number2" ]
then
echo "Number1 is equal to Number2"
else
echo "Number2 is greater than Number1"
fi
```

Toutes les méthodes ci-dessus peuvent activer le mode verbose.

Comment déboguer un script bash avec l'option xtrace (-x)

Le traçage d'exécution, également connu sous le nom de xtrace, est une option de débogage intelligente et utile, en particulier pour rechercher les erreurs logiques.

Les erreurs logiques sont généralement associées à des variables et des commandes.

Pour vérifier l'état d'une variable lors de l'exécution du script, nous utilisons l'option **-x**.

Maintenant, exécutez à nouveau le fichier **b_script.sh** avec l'indicateur **-x** :

```
...@...:~ $ bash -x /media/tmp/b_script.sh
+ echo 'Enter number1'
Enter number1
+ read number1
3
+ echo 'Enter number2'
Enter number2
+ read number2
5
+ '[' 3 -gt 5 ']'
+ '[' 3 -eq 5 ']'
+ echo 'Number2 is greater than Number1'
Number2 is greater than Number1
```

La sortie affiche explicitement la valeur de chaque variable pendant le processus d'exécution.

Encore une fois, **-x** peut être utilisé derrière le shebang ou après la ligne shebang en utilisant la commande set.

Xtrace place un signe + sur chaque ligne du script.

Comment déboguer un script bash avec l'option noexec (-n)

Les erreurs de syntaxe sont l'une des principales causes d'erreurs.

Pour le débogage syntaxique d'un script bash, nous utilisons le mode **noexec** (pas d'exécution).

Paramètre utilisé pour le mode noexec : **-n**.

Il n'affichera que les erreurs de syntaxe dans le code, sans l'exécuter.

Une approche beaucoup plus sûre du code de débogage.

Exécutez à nouveau b_script.sh avec l'option **-n** :

```
...@...:~ $ bash -n /media/tmp/b_script.sh
```

S'il n'y a pas d'erreur de syntaxe, le code ne sera pas exécuté.

Changeons maintenant notre code :

b_script.sh

```
#! /bin/bash
echo "Enter number1"
read number1
echo "Enter number2"
```

```
read number2
if [ "$number1" -gt "$number2" ]
then
echo "Number1 greater than number2"
elif [ "$number1" -eq "$number2" ]
#then
echo "Number1 is equal to Number2"
else
echo "Number2 is greater than Number1"
fi
```

Je commente le **then** après **elif**.

Maintenant exécutez le script **b_script.sh** avec **-n** :

```
...@...:~ $ bash -n /media/tmp/b_script.sh
/media/tmp/b_script.sh: ligne 12: erreur de syntaxe près du symbole
inattendu « else »
/media/tmp/b_script.sh: ligne 12: `else'
```

Comme prévu, il a clairement identifié l'erreur et l'a affichée dans le terminal.

Comment détecter les variables non définies lors du débogage d'un script bash

Une faute de frappe lors de l'écriture de code est une chose courante.

Souvent, vous entrez par erreur une variable, ce qui empêche l'exécution du code.

Pour détecter une telle erreur, nous utilisons l'option -u.

Changeons à nouveau le code :

```
#! /bin/bash
echo "Enter number1"
read number1
echo "Enter number2"
read number2
if [ "$num1" -gt "$number2" ]
then
echo "Number1 greater than number2"
elif [ "$number1" -eq "$number2" ]
then
echo "Number1 is equal to Number2"
else
echo "Number2 is greater than Number1"
fi
```

Tout d'abord, dans l'expression conditionnelle **if**, j'ai renommé la variable **number1** en **num1**.

Maintenant **num1** est une variable non définie.

Exécutons maintenant le script :

```
...@...:~ $ bash -u /media/tmp/b_script.sh
Enter number1
4
Enter number2
6
/media/tmp/b_script.sh: ligne 6: num1 : variable sans liaison
```

La sortie identifie et affiche explicitement le nom de la variable non définie.

Comment déboguer une partie spécifique d'un script bash

Le mode xtrace traite chaque ligne de code et produit un résultat.

Cependant, trouver des bogues dans un code volumineux prendra du temps si nous savons déjà quelle partie est potentiellement à l'origine du bogue.

Heureusement, xtrace vous permet également de déboguer un morceau de code spécifique, ce qui peut être fait avec la commande **installed**.

Placez **set -x** au début de la partie que vous devez déboguer, puis **set +x** à la fin.

Par exemple, pour déboguer les instructions conditionnelles de **b_script.sh** , enveloppez toutes les conditions entre **set -x** et **set +x** comme indiqué dans le code ci-dessous :

b_script.sh

```
#! /bin/bash
echo "Enter number1"
read number1
echo "Enter number2"
read number2
set -x
if [ "$number1" -gt "$number2" ]
then
echo "Number1 greater than number2"
elif [ "$number1" -eq "$number2" ]
then
echo "Number1 is equal to Number2"
else
echo "Number2 is greater than Number1"
fi
set +x
```

Maintenant lancez le script :

```
...@...:~ $ bash /media/tmp/b_script.sh
Enter number1
7
Enter number2
3
+ '[' 7 -gt 3 ']'
+ echo 'Number1 greater than number2'
Number1 greater than number2
+ set +x
```

La sortie est juste un débogage des conditions if comme indiqué.

Comment déboguer un script bash à l'aide de la commande trap

Si votre script est complexe, il existe des méthodes de débogage plus avancées.

L'une d'entre elles est la commande **trap**.

La commande **trap** capte les signaux et exécute la commande lorsqu'une situation spécifique se produit.

La commande peut être un signal ou une fonction.

J'ai créé un autre script appelé **sum_script.sh** :

sum_script.sh

```
#! /bin/bash
trap 'echo "Line ${LINENO}: first number is $number1, second
number is $number2, and sum is $sum" ' DEBUG
echo "Enter first number"
read number1
echo "Enter second number"
read number2
sum = $[number1 + number2]
echo "The sum is $sum"
```

La commande **trap** avec le signal **DEBUG** affiche l'état des variables **number1** , **number2** et **sum** après l'exécution de chaque ligne, comme illustré dans l'image de sortie suivante :

```
...@...:~ $ bash sum_script.sh
Line 3: first number is , second number is , and sum is
Enter first number
Line 4: first number is , second number is , and sum is
4
Line 5: first number is 4, second number is , and sum is
Enter second number
Line 6: first number is 4, second number is , and sum is
2
```

```
Line 7: first number is 4, second number is 2, and sum is
Line 8: first number is 4, second number is 2, and sum is 6
The sum is 6
```

Lignes 3, 4, 5, 6 et 7, les espaces après le mot “is” sont vides car l'utilisateur n'a pas encore entré de données ; ces espaces seront remplis au fur et à mesure que l'utilisateur saisira les valeurs.

Cette méthode est également très utile lors du débogage de scripts bash.

Comment déboguer un script bash sans substitution de fichier à l'aide de l'option -f

La substitution de fichiers est le processus de recherche de fichiers avec des caractères génériques, c'est-à-dire * et ?.

Dans de nombreux cas, vous n'avez pas besoin de développer les fichiers pendant le débogage.

Dans de tels cas, vous pouvez bloquer la substitution de fichiers avec l'option -f.

Voyons le script :

fglobe_script.sh

```
#! /bin/bash

echo "Display all text files"
ls *.txt
```

Le code ci-dessus affichera tous les fichiers texte du répertoire courant :

```
...@...:~ $ bash fglobe_script.sh
Display all text files
Doc1.txt  Doc2.txt  Doc3.txt
```

Pour désactiver la substitution de fichiers, utilisez l'option -f :

```
...@...:~ $ bash -f fglobe_script.sh
Display all text files
ls: impossible d'accéder à '*.txt': Aucun fichier ou dossier de ce type
```

De même, vous pouvez l'utiliser avec un shebang ou avec la commande set :

fglobe_script.sh

```
#! /bin/bash

set -f
```

```
echo "Display all text files"
ls *.txt
set +f
```

Exécutez maintenant bash fglobe_script.sh :

```
...@...:~ $ bash fglobe_script.sh
Display all text files
Doc1.txt  Doc2.txt  Doc3.txt
Display all text files
ls: impossible d'accéder à '*.txt': Aucun fichier ou dossier de ce type
```

Comment combiner les options de débogage pour déboguer un script shell

Nous n'utilisons qu'une seule option dans les méthodes de débogage ci-dessus, mais nous pouvons combiner différentes options pour une meilleure compréhension.

Implémentons les variantes **-x** et **-v** du script **sum_script.sh**. J'utilise le script **sum_script.sh** :

script.sh

```
#! /bin/bash
echo "Enter first number"
read number1
echo "Enter second number"
read number2
sum=$[number1 + number2]
echo "The sum is $sum"
```

Maintenant, lancez :

```
...@...:~ $ bash -xv sum_script.sh
#! /bin/bash
echo "Enter first number"
+ echo 'Enter first number'
Enter first number
read number1
+ read number1
4
echo "Enter second number"
+ echo 'Enter second number'
Enter second number
read number2
+ read number2
8
sum=$[number1 + number2]
+ sum=12
echo "The sum is $sum"
```

```
+ echo 'The sum is 12'
The sum is 12
```

Les sorties **-x** et **-v** sont combinées comme indiqué dans l'image de sortie.

De même, nous pouvons combiner l'option **-u** avec verbose -v pour détecter les erreurs.

Je remplace la variable number1 par num sur la sixième ligne du script :

sum_script.sh

```
#! /bin/bash
echo "Enter first number"
read number1
echo "Enter second number"
read number2
sum=$[num + number2]
echo "The sum is $sum"
```

Pour afficher le résultat, exécutez la commande suivante :

```
...@...:~ $ bash -uv sum_script.sh
#! /bin/bash
echo "Enter first number"
Enter first number
read number1
5
echo "Enter second number"
Enter second number
read number2
7
sum=$[num + number2]
sum_script.sh: ligne 6: num : variable sans liaison
```

Comment rediriger le rapport de débogage vers un fichier

L'enregistrement d'un rapport de débogage de script bash dans un fichier peut être utile dans de nombreuses situations.

C'est un peu délicat car rediriger le rapport de débogage vers un fichier ; nous utilisons des variables spéciales.

Implémentons-le sur le code b_script.sh :

```
#! /bin/bash
exec 5> debug_report.log
PS4='$LINENO-- '
```

```
BASH_XTRACEFD="5"
echo "Enter number1"
read number1
echo "Enter number2"
read number2
if [ "$number1" -gt "$number2" ]
then
echo "Number1 greater than number2"
elif [ "$number1" -eq "$number2" ]
then
echo "Number1 is equal to Number2"
else
echo "Number2 is greater than Number1"
fi
```

Dans la deuxième ligne de code, vous pouvez voir que nous redirigeons la sortie vers le fichier debug_report.log à l'aide de la commande exec avec le descripteur de fichier 5 (FD5).

- **exec 5> debug_report.log** : Dans exec , la commande redirige tout ce qui se passe dans le shell vers le fichier debug_report.log.
- **BASH_XTRACEFD = 5** : variable spécifique à bash qui ne peut être utilisée dans aucun autre shell. Un descripteur de fichier valide doit lui être attribué et bash écrira la sortie extraite dans debug_report.log.
- **PS4='$LINENO– '** : variable bash utilisée pour imprimer le numéro de ligne lors du débogage en mode xtrace. PS4 par défaut : signe +

Le script ci-dessus crée un fichier journal nommé debug_report.log, pour le lire, utilisez la commande cat :

```
...@...:~ $ bash -x b_script.sh
+ exec
+ PS4='$LINENO-- '
4-- BASH_XTRACEFD=5
Enter number1
4
Enter number2
5
Number2 is greater than Number1
...@...:~ $  cat debug_report.log
5-- echo 'Enter number1'
6-- read number1
7-- echo 'Enter number2'
8-- read number2
9-- '[' 4 -gt 5 ']'
12-- '[' 4 -eq 5 ']'
16-- echo 'Number2 is greater than Number1'
```

Pré-requis

Première étape

Autres étapes

Conclusion

Un code plein d'erreurs peut affecter les performances du programme ou nuire au matériel.
Le débogage est très important pour chaque programme car il le rend plus efficace.
La détection de bogues existants et potentiels pendant le développement du programme peut empêcher un comportement inattendu dans votre programme.
Les codes volumineux nécessitent généralement un débogage actif, ce qui améliore l'efficacité du code en éliminant les fragments de code consommateurs de ressources.

De nombreux langages de programmation et frameworks ont leurs propres débogueurs compagnons.
Les scripts bash peuvent implémenter diverses méthodes de débogage de script.
Ce guide détaille toutes les méthodes qui peuvent être utilisées pour trouver des erreurs dans les scripts bash.
Donc, chaque fois que vous sentez que votre script bash ne fonctionne pas comme prévu, utilisez l'une des méthodes mentionnées ci-dessus, mais dans la plupart des cas, le mode xtrace (-x) est très utile.

Problèmes connus

Voir aussi

- **(ru)** https://softoban.com/how-debug-bash-script

Basé sur « Comment déboguer un script bash ? » par Auteur.