eBook Gratuit

# APPRENEZ batch-file

eBook gratuit non affilié créé à partir des **contributeurs de Stack Overflow.**

#batch-file

# Table des matières

# À propos

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: batch-file

It is an unofficial and free batch-file ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official batch-file.



Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

# Chapitre 1: Démarrer avec le fichier de commandes

## Remarques

De Microsoft Technet:

> Avec les fichiers de commandes, également appelés programmes par lots ou scripts, vous pouvez simplifier les tâches routinières ou répétitives. Un fichier de commandes est un fichier texte non formaté qui contient une ou plusieurs commandes et possède une extension de nom de fichier .bat ou .cmd. Lorsque vous tapez le nom de fichier à l'invite de commandes, Cmd.exe exécute les commandes de manière séquentielle, telles qu'elles apparaissent dans le fichier.

**Noms et extensions des fichiers batch**

| Extension | Remarques |
|---|---|
| .chauve souris | Cette extension s'exécute avec MS-DOS et toutes les versions de Windows |
| .cmd | Utilisé pour les fichiers de commandes dans la famille Windows NT |
| .btm | L'extension utilisée par 4DOS et 4NT |

Pour comprendre la différence entre `.cmd` et `.bat` veuillez voir ici .

Évitez les noms qui sont déjà le nom de commandes intégrées. comme `tracert` . Il existe un utilitaire appelé `tracert.exe` . Donc, évitez de nommer un fichier de commandes `tracert.bat`

**Fichier de commandes en cours d'exécution**

La manière la plus simple d'exécuter un fichier de commandes est de double-cliquer sur son icône. Ou collez le chemin complet du fichier dans une invite de commande, ou simplement son nom si la commande Invite a été lancée à partir du répertoire du fichier de commandes, puis entrez.

Exemple:

```
C:\Foo\Bar>test.bat
C:\Foo\Bar>C:\Foo\Bar\Baz\test.bat
```

## Examples

### Ouvrir une invite de commandes

L'invite de commande est préinstallée sur tous les systèmes d'exploitation Windows NT, Windows CE, OS / 2 et eComStation et existe sous la forme `cmd.exe` , généralement située dans `C:\Windows\system32\cmd.exe`

Sous Windows 7, les moyens les plus rapides d'ouvrir l'invite de commande sont les suivants:

- Appuyez sur `Win` , tapez «cmd», puis appuyez sur `Entrée` .
- Appuyez sur `Win` + `R` , tapez "cmd" puis appuyez sur `Entrée` .
- Si vous avez une fenêtre d'explorateur ouverte, tapez "cmd" dans la barre d'adresse pour ouvrir une invite dans le répertoire actuellement sélectionné.
- Faites un clic droit sur un dossier dans l'Explorateur tout en maintenant la `touche Maj enfoncée` et sélectionnez "Ouvrir la fenêtre de commande ici".

Vous pouvez également l'ouvrir en naviguant vers l'exécutable et en double-cliquant dessus.

Dans certains cas, vous devrez peut-être exécuter `cmd` avec des autorisations élevées. Dans ce cas, cliquez avec le bouton droit de la souris et sélectionnez "Exécuter en tant qu'administrateur". Cela peut également être réalisé en appuyant sur `Ctrl` + `Maj` + `Entrée` au lieu de `Entrer` en utilisant la voie 1 des points ci-dessus.

## Modification et affichage de fichiers par lots

Tout éditeur ASCII peut éditer des fichiers batch. Une liste des éditeurs qui peuvent utiliser la syntaxe pour mettre en évidence la syntaxe de lot peut être trouvée ici . Vous pouvez également utiliser le bloc-notes par défaut fourni avec Windows pour modifier et afficher un fichier de commandes, même s'il ne propose pas de mise en évidence de la syntaxe.

Pour ouvrir le bloc-notes:

- Appuyez sur `Win` + `R` , tapez `notepad` , puis appuyez sur `Entrée` .

Sinon, le moyen le plus "primitif" de créer un fichier de commandes est de rediriger la sortie de la ligne de commande vers un fichier, par exemple.

```
echo echo hello world > first.bat
```

qui écrit `echo hello world` dans le fichier `first.bat` .

Vous pouvez modifier un fichier de commandes en cliquant avec le bouton droit sur le fichier et en sélectionnant "Modifier" dans le menu contextuel.

Pour afficher le contenu d'un fichier de commandes à partir d'une invite de commandes, exécutez la commande suivante:

```
type first.bat
```

Vous pouvez également commencer à éditer votre fichier de commandes avec notepad à partir de

l'invite de commande en tapant

```
notepad first.bat
```

## Obtenir de l'aide

Pour obtenir de l'aide sur une commande de fichier de commandes, vous pouvez utiliser l'aide intégrée.

Ouvrez une invite de commande (dont l'exécutable est `cmd.exe` ) et entrez `help` pour voir toutes les commandes disponibles.

Pour obtenir de l'aide sur l'une de ces commandes, tapez `help` suivi du nom de la commande.

Par exemple:

```
help help
```

Affichera:

```
Provides help information for Windows commands.

HELP [command]

    command - displays help information on that command.
```

Certaines commandes afficheront également de l'aide si elles sont suivies de `/?` .

Essayer:

```
help /?
```

Remarque:

`Help` n'affichera que l'aide pour **les** commandes **internes** .

Lire Démarrer avec le fichier de commandes en ligne: https://riptutorial.com/fr/batch-file/topic/1277/demarrer-avec-le-fichier-de-commandes

# Chapitre 2: Ajouter un délai au fichier batch

## Introduction

Cette rubrique vous apprendra une des nombreuses choses utiles à connaître dans le langage de script, le fichier batch; Ajout d'un délai / pause / délai d'attente à votre fichier de commandes.

## Examples

### Temps libre

---

## Temps libre

La manière la plus simple de retarder ou de suspendre un certain temps est d'utiliser la commande standard `TIMEOUT` . Pour faire un délai qui dure exactement une minute, tapez:

```
timeout /t 60
```

Maintenant qu'est-ce qui se passe ici?

Tout d'abord, nous utilisons la commande `TIMEOUT` avec le paramètre `/T` (qui signifie simplement timeout), puis nous spécifions la quantité de secondes à attendre. Dans ce cas ... `60` secondes.

**Délai d'attente avec le paramètre / NOBREAK**

Si nous prenons l'exemple d'avant et que nous l'exécutons dans un fichier BATCH: `timeout /t 60` alors que vous attendez ces 60 secondes, vous pouvez réellement interrompre le délai en appuyant sur n'importe quelle touche de votre clavier. Pour éviter cela, nous ajoutons simplement le paramètre `/NOBREAK` à la fin.

```
timeout /t 60 /nobreak
```

En faisant cela, il expirera pendant 60 secondes et si vous souhaitez interrompre le délai, vous devrez appuyer sur (CTRL-C) sur votre clavier.

**Délai d'attente silencieux**

Quand il fait un timeout, il affichera:

```
Waiting for X seconds, press a key to continue ...
or
Waiting for X seconds, press CTRL+C to quit ... [This is with the /NOBREAK parameter]
```

Pour masquer le message, utilisez l'argument `NUL` (pour l'explication de `NUL` : **cliquez ici** )

```
timeout /t 60 > nul
or
timeout /t 60 /nobreak > nul
```

### Pause

Pour suspendre votre script, utilisez simplement la commande `PAUSE` .

```
PAUSE
```

Cela affichera le texte `Press any key to continue . . .` , puis ajoutez une nouvelle ligne à la saisie de l'utilisateur.

Disons que nous voulons créer un programme "Hello World" et après avoir cliqué sur quelque chose sur notre clavier, nous voulons qu'il quitte le programme avec la commande `EXIT` .

```
echo Hello World
pause
exit
```

Ici, il utilise la commande `ECHO` pour dire "Hello World". Ensuite, nous utilisons la commande `PAUSE` qui affiche `Press any key to continue . . .` et ensuite nous utilisons la commande `EXIT` pour terminer le script BATCH en cours.

Quand il fait une pause, il affiche:

```
Press any key to continue . . .
```

#### Masquer la "Appuyez sur une touche pour continuer ... invite

Pour masquer le message, nous redirigeons la sortie vers un périphérique spécial appelé `nul` . Ce n'est pas vraiment un appareil réel, mais tout ce que nous lui envoyons est jeté.

```
pause > nul
```

### Ping

---

## Ping

L'une des commandes les plus utilisées pour retarder un certain temps est la commande `ping` .

#### Utilisation de base

```
PING -n 1 -w 1000 1.1.1.1

REM the -n 1 flag means to send 1 ping request.
REM the -w 1000 means when the IP(1.1.1.1) does not respond, go to the next command
```

```
REM 1.1.1.1 is an non-existing IP so the -w flag can ping a delay and go to next command
```

Cela produirait ce qui suit sur votre fichier de commandes / console:

```
C:\Foo\Bar\Baz>ping -n -w 1000 1.1.1.1

Pinging 1.1.1.1 (Using 32 bytes of data)
Request timed out

Ping statistics for 1.1.1.1
    Packets: Sent = 2,Received = 0, Lost = 1(100% loss)
```

### Masquer le texte fait écho

Ajoutez simplement `>nul` à la fin de la commande pour le rediriger vers null.

```
ping -n w 1000 1.1.1.1 >nul
```

Cela ne produirait rien.

### Dormir

---

## Dormir

Sur les anciens systèmes Windows, le `timeout` n'est pas disponible. Cependant, nous pouvons utiliser la commande `sleep` .

### Usage

```
sleep 1
```

Très explicite dormir pendant 1 seconde. Cependant, le `sleep` est une commande désespérée et doit être remplacé par un délai d' attente .

### Disponibilité

Cette commande est disponible sur l'ancien système Windows. `SLEEP.exe` est également inclus dans le Kit de ressources 2003.

Pour utiliser `sleep.exe` , placez le fichier exécutable dans le dossier `%Windir%\System32` . Ensuite, vous pouvez l'utiliser comme commande normale.

Lire Ajouter un délai au fichier batch en ligne: https://riptutorial.com/fr/batch-file/topic/9123/ajouter-un-delai-au-fichier-batch

# Chapitre 3: Arguments de ligne de commande de fichier batch

## Examples

### Arguments de ligne de commande fournis aux fichiers de commandes

Les arguments de ligne de commande du fichier batch sont des valeurs de paramètres soumises lors du démarrage du lot. Ils doivent être placés entre guillemets s'ils contiennent des espaces. Dans un fichier batch en cours d'exécution, les arguments sont utilisés à des fins diverses, par exemple la redirection vers `:labels` , les variables de configuration ou les commandes en cours d'exécution.

Les arguments sont mentionnés dans le fichier de commandes à l'aide de `%1, %2, ..., %9` .

```
@echo off
setlocal EnableDelayedExpansion
if not "%1"=="" (
    set "dir=%~1" & set "file=%~2"
    type !dir!\!file! | find /n /i "True" >nul^
     && echo Success! || echo Failure
)
exit /b

C:\Users\UserName> test.bat "C:\Temp\Test Results" "Latest.log"
Success!
```

Remarques:

- Dans l'exemple ci-dessus, les guillemets doubles sont supprimés en utilisant le modificateur d'argument `%~1` .
- Les longues chaînes sont divisées en plusieurs lignes à l'aide de `^` et il y a un espace avant le caractère sur la ligne suivante.

### Fichiers batch avec plus de 9 arguments

Lorsque plus de 9 arguments sont fournis, la commande `shift [/n]` peut être utilisée, où `/n` signifie commencer au nième argument, n est compris entre zéro et huit.

**En boucle à travers les arguments:**

```
:args
set /a "i+=1"
set arg!i!=%~1
call echo arg!i! = %%arg!i!%%
shift
goto :args
```

Notez que dans l'exemple ci-dessus, la variable d'extension retardée `i` est utilisée pour affecter des valeurs d'argument au tableau de variables. La commande `call` permet d'afficher ces valeurs de variables dans la boucle.

**Arguments de comptage:**

```
for %%i in (%*) do (set /a ArgCount+=1)
echo %ArgCount%
```

**Définissez une variable sur son argument:**

```
set i=5
call set "path%i%=%%~i"
```

## Déplacement des arguments entre parenthèses

Ayons l' `example.bat` suivant et appelons-le avec les arguments `1` , `2` et `3` :

```
@echo off

(
    shift
    shift
    echo %1
)
```

Comme l'expansion de la variable changera une fois que le contexte des parenthèses de fin sera atteint, la sortie sera:

    1

Comme cela pourrait poser problème lors du changement de parenthèse pour accéder à l'argument, vous devrez utiliser l'appel:

```
@echo off

(
    shift
    shift
    call echo %%1
)
```

maintenant la sortie sera `3` . Comme la commande `CALL` est utilisée (ce qui entraînera une extension de variable supplémentaire) avec cette technique, les arguments accédant peuvent également être paramétrés:

```
@echo off

set argument=1

    shift
```

```
    shift
    call echo %%%argument%
```

avec expansion retardée:

```
@echo off
setlocal enableDelayedExpansion
set argument=1

    shift
    shift
    call echo %%!argument!
```

la sortie sera

    3

Lire Arguments de ligne de commande de fichier batch en ligne: https://riptutorial.com/fr/batch-file/topic/4981/arguments-de-ligne-de-commande-de-fichier-batch

# Chapitre 4: Bogues dans le processeur cmd.exe

## Introduction

Cette rubrique se concentrera sur les erreurs causées par les bogues du processeur. Voici les éléments sur lesquels nous nous concentrerions sur la cause et la solution du problème.

## Remarques

Dans l'exemple DEL File Extension , l'utilisateur X. Liu remarque que ce bogue ne **se** produit **pas** lorsque l'extension de fichier dans la commande `DEL` est inférieure à 3 caractères.

## Examples

### Parenthèses confusion

À partir de ce site Web, le PO a remarqué un problème.

---

# Cause

Considérez l'extrait de code suivant.

```
if 1==1 (
    set /a result = 2*(3+4)
)
```

À première vue, vous pensez peut-être que `CMD.exe` le traiterait comme `CMD.exe` :

- La condition est vraie, exécutez le bloc de code
- Définir la valeur du résultat de la variable sur `14`
- Continuer

Cependant, le processus `CMD.exe` comme tel:

- La condition est vraie, exécutez le bloc de code
- **Calculer `2*(3+4` , the `)` après que 4 soit traité à la fin du bloc de code `if`**
- Un hasard `)` est apparu!

La deuxième étape renverrait une erreur de `Unbalanced parentheses` .

---

# Solution

Selon un ensemble allemand CMD.exe `set /?` , nous devrions citer des opérations arithmétiques. Voici un exemple.

| précédent | Résultat |
| --- | --- |
| `set /a result=2+5*4` | `set /a result="2+5*4"` |

Par ailleurs, selon un ensemble anglais CMD.exe `set /?` , des guillemets sont requis si des opérateurs logiques ou modulaires sont présents dans l'expression (bien que ce ne soit pas une étape *indispensable* ).

### Caractère d'évasion inapproprié

Dans cette question , l'utilisateur txtechhelp a détecté un problème avec le caractère `^` qui pouvait poser un problème de sécurité.

---

# Cause

```
anyInvaildCommand ^
```

Note: Assurez-vous que le curseur ( `^` ) est le dernier caractère! Tout `CR\LF` supplémentaire ne fonctionnera pas du tout!

Le caret cherche le prochain personnage à échapper. Cependant, les caractères ne sont plus disponibles pour s'échapper, donc `cmd` boucle à l'infini, à la recherche d'un personnage à échapper. Dans ce processus de "boucle", `cmd.exe` consommera la mémoire de votre ordinateur. Et manger graduellement toute la mémoire, mettant l'ordinateur à genoux.

Ce problème peut entraîner des problèmes de sécurité plus graves, car il suffit de saisir le code sur l'ordinateur déverrouillé.

# Solutions

- Utiliser la page de codes **UTF-16** pourrait résoudre ce problème. Seul **UTF-8** ou **ASCII** provoquerait le bogue.
- Assurez-vous qu'il y a un `CR\LF` supplémentaire dans le fichier, ou simplement n'utilisez pas caret à la fin du fichier.

# Supplémentaire

Ce bug semble être résolu dans Windows 10.

### Extension de fichier DEL

Ce bogue a été signalé par steve2916 à partir de ce fil de discussion Microsoft Windows.

---

# Cause

Considérons un dossier avec de tels fichiers.

```
TestA.doc
TestB.doc
TestC.docx
TestD.docx
```

Si nous voulons supprimer tous les fichiers `.doc` de ce répertoire, nous ferions généralement ce qui suit:

```
del *.doc
```

Cependant, cette commande supprime également les fichiers `.docx` . La même chose se produit sur les extensions de fichiers avec ce modèle.

| Déposer un | Fichier B |
| --- | --- |
| N'importe quel nom. **abc** | Un autre nom. **ab** d |

Comme nous pouvons le voir, tant que la chaîne d'extension de fichier contient la chaîne utilisée dans la commande `del` , le fichier sera supprimé. Notez que ce bogue se produit uniquement lorsque la chaîne d'extension utilisée dans la commande `del` contient au moins trois caractères. Par exemple, `del *.do` ne supprime pas `A.doc` ou `A.docx` .

# Solution

Dans le sujet, l'utilisateur MicroCompsUnltd a remarqué que l'utilisation de `Powershell` résoudrait le problème.

Lire Bogues dans le processeur cmd.exe en ligne: https://riptutorial.com/fr/batch-file/topic/10694/bogues-dans-le-processeur-cmd-exe

# Chapitre 5: Changer de répertoire et lister son contenu

## Syntaxe

- echo% cd% - affiche le chemin actuel du répertoire
- cd "C: \ chemin \ vers \ un \ répertoire" - change le chemin du répertoire
- cd "% variable_containing_directory_path%" - modifie également le chemin du répertoire
- cd / d E: - changer en E: lecteur d'un lecteur différent
- cd / - change le répertoire pour revenir au lecteur actuel
- echo `%__CD__%` - affiche le chemin d'accès actuel du répertoire avec une barre oblique inverse (non documentée)
- echo% = C:% - Le répertoire actuel du lecteur C: (non documenté)
- echo% = D:% - Le répertoire actuel du lecteur D: si le lecteur D: a été accédé dans la session CMD en cours (non documenté)

## Remarques

Pourquoi est-ce important et quelles sont leurs utilisations et avantages:

- ouvrir un fichier ou une application dans un répertoire à l'aide d'un lot
- créer et écrire et lire des fichiers dans un répertoire à l'aide d'un lot
- connaître et lister tous les dossiers
- savoir où votre fichier batch est en cours d'exécution

## Examples

### Pour afficher le répertoire en cours

Format et utilisation:

```
echo %cd%
```

`%cd%` est une variable système contenant le chemin du répertoire en cours

### Pour changer le répertoire actuel (sans changer de lecteur)

Format:

```
cd "<path>"
```

Exemple:

```
 cd "C:\Program Files (x86)\Microsoft Office"
```

`cd` est une abréviation de `chdir` et les deux commandes se comportent exactement de la même manière. Par souci de cohérence, `cd` sera utilisé tout au long de ce sujet.

---

Pour accéder au répertoire d' un niveau supérieur au répertoire courant, spécifiez le répertoire système `. . `.

```
cd ..
```

---

Pour accéder à un répertoire situé à l'intérieur du répertoire actuel, `cd` simplement au nom du dossier sans saisir le chemin complet (en entourant le nom du répertoire entre guillemets s'il contient des espaces).

Par exemple, pour entrer `C:\Program Files (x86)\Microsoft Office` dans le répertoire `C:\Program Files (x86)` , la syntaxe suivante peut être utilisée:

```
cd "Microsoft Office"
```

ou

```
cd "C:\Program Files (x86)\Microsoft Office"
```

## Navigation dans un répertoire sur un autre lecteur

`cd` lui-même ne permettra pas à un utilisateur de se déplacer entre les lecteurs. Pour passer à un autre lecteur, l'option `/d` doit être spécifiée.

Par exemple, passer de `C:\Users\jdoe\Desktop` à `D:\Office Work`

```
cd /d "D:\Office Work"
```

## Comment afficher tous les dossiers et fichiers dans un répertoire

Utilisation pour répertorier tous les dossiers et fichiers du répertoire en cours: `dir`

Un répertoire cible peut également être spécifié: `dir C:\TargetPath`

Lorsque vous spécifiez un chemin avec des espaces, il doit être entouré de guillemets: `dir "C:\Path With Spaces"`

## Changer de lecteur sans CD / D

```
Pushd "D:\Foo"
Dir
Popd
```

`Pushd` changera le répertoire en répertoire suivant (dans ce cas D: \ Foo. `Popd` retourne au répertoire original.

### Pour changer le répertoire en cours à la racine du lecteur en cours

Format:

```
cd/
```

`cd/` est défini pour changer le répertoire en cours à la racine du lecteur en cours

Lire Changer de répertoire et lister son contenu en ligne: https://riptutorial.com/fr/batch-file/topic/7132/changer-de-repertoire-et-lister-son-contenu

# Chapitre 6: Commandes batch obsolètes et leurs remplacements

## Examples

### DÉBOGUER

`DEBUG` commande `DEBUG` été utilisée pour créer des fichiers binaires / exécutables à partir d'un fichier de commandes. La commande est toujours disponible dans les versions 32 bits de Windows, mais elle est capable de créer des fichiers binaires uniquement avec des instructions 16 bits, ce qui la rend inutilisable sur les machines 64 bits. Maintenant, CERTUTIL est utilisé à cet effet, ce qui permet de décoder / encoder des fichiers binaires / médias à partir de formats HEX ou BASE64. Exemple avec un fichier qui crée un fichier icône:

```
@echo off

del /q /f pointer.jpg >nul 2>nul
certutil -decode "%~f0" pointer.jpg
hh.exe pointer.jpg
exit /b %errorlevel%

-----BEGIN CERTIFICATE-----
/9j/4AAQSkZJRgABAgAAZABkAAD/7AARRHVja3kAAQAEAAAAMgAA/+4ADkFkb2Jl
AGTAAAAAAf/bAIQACAYGBgYGCAYGCAwIBwgMDgoICAoOEA0NDg0NEBEMDg0NDgwR
DxITFBMSDxgYGhoYGCMiIiIjJycnJycnJycnJwEJCAgJCgkLCQkLDgsNCw4RDg4O
DhETDQ0ODQ0TGBEPDw8PERgWFxQUFBcWGhoYGBoaISEgISEnJycnJycnJycn/8AA
EQgACgAKAwEiAAIRAQMRAf/EAFsAAQEBAAAAAAAAAAAAAAAAAAAAAAAGBwEBAQAAAAAA
AAAAAAAAAAAAAAEQAAIBAwQDAAAAAAAAAAAAAAAAEDAgARBSExIwQSIhMRAQEBAAAA
AAAAAAAAAAAAAAAAARIf/aAAwDAQACEQMRAD8A13PZ5eIX3gO8ktKZfFPksvQ8r4uL
ecJmx1BcWGhoYGBoaISEgISEnJycnJycnJycn/9k=
-----END CERTIFICATE-----
```

Voici la même chose avec le format hexadécimal:

```
@echo off

(echo 0000    ff d8 ff e0 00 10 4a 46  49 46 00 01 02 00 00 64)   >pointer.hex
(echo 0010    00 64 00 00 ff ec 00 11  44 75 63 6b 79 00 01 00)   >>pointer.hex
(echo 0020    04 00 00 00 32 00 00 ff  ee 00 0e 41 64 6f 62 65)  >>pointer.hex
(echo 0030    00 64 c0 00 00 00 01 ff  db 00 84 00 08 06 06 06)   >>pointer.hex
(echo 0040    06 06 08 06 06 08 0c 08  07 08 0c 0e 0a 08 08 0a)   >>pointer.hex
(echo 0050    0e 10 0d 0d 0e 0d 0d 10  11 0c 0e 0d 0d 0e 0c 11)   >>pointer.hex
(echo 0060    0f 12 13 14 13 12 0f 18  18 1a 1a 18 18 23 22 22)   >>pointer.hex
(echo 0070    22 23 27 27 27 27 27 27  27 27 27 27 01 09 08 08)   >>pointer.hex
(echo 0080    09 0a 09 0b 09 09 0b 0e  0b 0d 0b 0e 11 0e 0e 0e)   >>pointer.hex
(echo 0090    0e 11 13 0d 0d 0e 0d 0d  13 18 11 0f 0f 0f 0f 11)   >>pointer.hex
(echo 00a0    18 16 17 14 14 14 17 16  1a 1a 18 18 1a 1a 21 21)   >>pointer.hex
(echo 00b0    20 21 21 27 27 27 27 27  27 27 27 27 27 ff c0 00)   >>pointer.hex
(echo 00c0    11 08 00 0a 00 0a 03 01  22 00 02 11 01 03 11 01)   >>pointer.hex
(echo 00d0    ff c4 00 5b 00 01 01 01  00 00 00 00 00 00 00 00)   >>pointer.hex
(echo 00e0    00 00 00 00 00 00 06 07  01 01 01 00 00 00 00 00)   >>pointer.hex
(echo 00f0    00 00 00 00 00 00 00 00  00 00 01 10 00 02 01 03)   >>pointer.hex
```

```
(echo 0100    04 03 00 00 00 00 00 00  00 00 00 00 01 03 02 00)   >>pointer.hex
(echo 0110    11 05 21 31 23 04 12 22  13 11 01 01 01 00 00 00)   >>pointer.hex
(echo 0120    00 00 00 00 00 00 00 00  00 00 00 11 21 ff da 00)   >>pointer.hex
(echo 0130    0c 03 01 00 02 11 03 11  00 3f 00 d7 73 d9 e5 e2)   >>pointer.hex
(echo 0140    17 de 03 bc 92 d2 99 7c  53 e4 b2 f4 3c af 8b 8b)   >>pointer.hex
(echo 0150    79 c2 66 c7 50 4c 49 b9  bc 0f a5 15 29 57 20 ff)   >>pointer.hex
(echo 0160    d9                                                  )   >>pointer.hex

certutil -decodehex "pointer.hex" pointer.jpg
hh.exe pointer.jpg
exit /b %errorlevel%
```

Comme vous pouvez le voir, hex nécessite un fichier temporaire supplémentaire et les extensions de format hexadécimal sont plus grandes

## AJOUTER

`APPEND` était une commande dans msdos qui permettait d'utiliser des fichiers de ressources / médias tels qu'ils sont dans le même répertoire. La commande est toujours disponible dans les versions 32 bits de Windows mais ne semble pas fonctionner. Dans certaines sources (y compris les microsofts), il est indiqué que la commande est remplacée par DPATH, mais ce n'est pas tout à fait vrai. Malgré le message d'aide DPATH qui pointe vers la commande APPEND, sa syntaxe est identique à celle de `PATH` . Les répertoires listés dans DPATH peuvent être utilisés avec une redirection d'entrée ou une commande de type :

```
@echo off
dpath %windir%

set /p var=<win.ini
echo using dpath with input redirection:
echo %var%
echo.
echo using dpath with type command:
type win.ini
```

## BITSADMIN

`BITSADMIN` été utilisé pour transférer des documents, télécharger un site Web, télécharger des fichiers à partir de sites Web, etc. Cette commande est une commande obsolète et peut être supprimée sur les prochaines mises à jour Windows. Pour éviter ce problème, utilisez la nouvelle `BIT cmdlet` Powershell `BIT cmdlet` .

---

Voici un exemple de code utilisant `BITSADMIN` .

```
@echo off
Bitsadmin /create /download Stackoverflow.com
rem download the website StackOverflow.com
```

Lire Commandes batch obsolètes et leurs remplacements en ligne: https://riptutorial.com/fr/batch-file/topic/10109/commandes-batch-obsoletes-et-leurs-remplacements

# Chapitre 7: Commentaires dans les fichiers batch

## Introduction

Les commentaires sont utilisés pour afficher des informations dans un script de commandes.

## Syntaxe

- REM
- & REM
- ::
- & ::
- Aller à: Label

```
    Comments. You can also use |>< ,etc.
```

:Étiquette

## Examples

### Utiliser REM pour les commentaires

```
REM This is a comment
```

- `REM` est la commande de commentaire officielle.

### Utiliser les étiquettes comme commentaires

```
::This is a label that acts as a comment
```

Le double-colon `::` comment montré ci-dessus n'est pas documenté comme étant une commande de commentaire, mais il s'agit d'un cas spécial d'étiquette qui agit comme un commentaire.

**Attention** : lorsque des étiquettes sont utilisées comme commentaires dans un bloc de code entre crochets ou `for` commande, le processeur de commandes attend que chaque étiquette soit suivie par au moins une commande.

Le shell `cmd` essaiera d'exécuter la deuxième ligne même si elle est formatée comme une étiquette (et **cela provoque une erreur** ):

```
(
echo This example will fail
:: some comment
)
```

Lorsque vous travaillez avec des blocs de code entre parenthèses, il est plus sûr d'utiliser REM pour toutes les lignes de commentaires.

## Utiliser les variables comme commentaires

Il est également possible d'utiliser des variables en tant que commentaires. Cela peut être utile pour empêcher conditionnellement l'exécution des commandes:

```
@echo off
setlocal
if /i "%~1"=="update" (set _skip=) Else (set _skip=REM)
%_skip% copy update.dat
%_skip% echo Update applied
...
```

Lorsque vous utilisez l'extrait de code ci-dessus dans un fichier de commandes, les lignes commençant par `%_skip%` ne sont exécutées que si le fichier de commandes est appelé avec la `update` à `update` tant que paramètre.

## Bloquer les commentaires

Le format de fichier de commandes ne comporte pas de syntaxe de commentaire de bloc, mais il existe une solution simple à ce problème.

Normalement, chaque ligne d'un fichier de commandes est lue et ensuite exécutée par l'analyseur, mais une `goto` peut être utilisée pour sauter un bloc de texte brut (qui peut être utilisé comme un commentaire de bloc):

```
@echo off
goto :start

A multi-line comment block can go here.
It can also include special characters such as | >

:start
```

Puisque l'analyseur ne voit jamais les lignes entre l' `goto :start` et `:start` label, il peut contenir du texte arbitraire (y compris des caractères de contrôle sans avoir besoin de les échapper) et l'analyseur ne générera pas d'erreur.

## Commenter la ligne du code

Pour commenter la même ligne que le code, vous pouvez utiliser `&::` ou `&rem` . Vous pouvez également utiliser `&&` ou `||` pour remplacer `&` .

Exemple :

```
@echo off
echo This is a test &::This is a comment
echo This is another test &rem This is another comment
pause
```

**Une curiosité** : la commande `SET` permet des commentaires en ligne *limités* sans `&rem` :

```
set "varname=varvalue"    limited inline comment here
```

Limites:

- syntaxe avec guillemets doubles `set "varname=varvalue"` ou `set "varname="` ,
- un commentaire en ligne **ne peut** contenir aucun double guillemet,
- tous les personnages toxiques `cmd | < > &` **doit être** correctement échappé comme `^| ^< ^> ^&` ,
- les parenthèses `( )` **doivent être** correctement échappées en tant que `^( ^)` dans un bloc de code entre crochets.

## Lot et WSF Hybrid Commentaire

```
<!-- : Comment
```

Cela fonctionne à la fois avec le script batch et WSF. La balise de fermeture ( `-->` ) ne fonctionne que dans WSF.

| Code | Sucessful dans les deux lots et WSF? |
|---|---|
| `<!--: Comment` | Vrai |
| `<!--: Comment -->` | False - La balise de fermeture ne fonctionne que pour WSF |
| `-->` | Faux |

Lire Commentaires dans les fichiers batch en ligne: https://riptutorial.com/fr/batch-file/topic/3152/commentaires-dans-les-fichiers-batch

# Chapitre 8: Contourner les limitations arithmétiques dans les fichiers de commandes

## Introduction

Les fichiers par lots ne permettent que des calculs sur des nombres entiers de 32 bits, bien que cela puisse être contourné par différentes approches.

## Examples

### Utiliser Powershell

Comme le PowerShell est installé par défaut sur tous les systèmes Windows à partir de 7/2008, il peut être utilisé pour des calculs plus complexes:

```
@echo off
set "expression=(2+3)*10/1000"
for /f %%# in ('"powershell %expression%"') do set result=%%#
echo %result%
```

Attention aux doubles guillemets supplémentaires dans le `for /f` qui empêchent les crochets d'entrer en conflit avec la syntaxe de la commande for.

Le problème potentiel est que Powershell est beaucoup plus lent que l'utilisation de wsh / vbscript / jscript à cause du chargement du framerwork .net

### Utiliser jscript

`WSH/JScript` est installé sur tous les systèmes Windows depuis que l'utilisation de NT pour des calculs plus complexes la rend très portable. JScript est plus facile à combiner avec un fichier batch:

```
@if (@codesection==@batch) @then
@echo off

set "expression=2*(2+3)/1000"
for /f %%# in ('cscript //nologo //e:jscript "%~f0" "%expression%"') do set
result=%%#
echo %result%
:: more batch code

exit /b %errorlevel%
@end
WScript.Echo(eval(WScript.Arguments(0)));
```

Avec cette approche, vous pouvez mettre tout votre code dans un seul fichier. C'est plus rapide que d'utiliser Powershell. Ici et ici, des scripts plus avancés peuvent être trouvés (qui peuvent être utilisés comme fichiers externes).

## Emulation de calculs de stylo et de papier, implémentations de fonctions mathématiques

1. ***On*** peut trouver ici la bibliothèque mathématique la plus complète qui émule les calculs de stylo et de papier et qui permet de travailler avec des nombres plus importants.
2. Voici d'autres exemples d'émulations de stylo et de papier: ADD , Comparison , Multiply
3. Certaines implémentations de fonctions mathématiques peuvent être trouvées ici .

Lire Contourner les limitations arithmétiques dans les fichiers de commandes en ligne: https://riptutorial.com/fr/batch-file/topic/10702/contourner-les-limitations-arithmetiques-dans-les-fichiers-de-commandes

# Chapitre 9: Création de fichiers à l'aide de lots

## Introduction

Une fonctionnalité utile des fichiers de commandes est la possibilité de créer des fichiers avec eux. Cette section montre comment créer des fichiers en utilisant un code de lot.

## Syntaxe

- echo (tapez ici ce que vous voulez dans le to be) >> (filename)
- echo (nom de la variable) >> (nom du fichier)

## Remarques

Si un fichier existe, `>` écrasera le fichier et `>>` s'ajoutera à la fin du fichier. Si un fichier n'existe pas, les deux créeront un nouveau fichier.

De plus, la commande `echo` ajoute automatiquement une nouvelle ligne après votre chaîne.

Alors

```
echo 1 > num.txt
echo 1 > num.txt
echo 2 >> num.txt
```

va créer le fichier suivant:

```
1
2
```

Pas ça:

```
1 1 2
```

ou

```
1 2
```

De plus, vous ne pouvez pas simplement modifier une seule ligne dans un fichier texte. Vous devez lire le fichier en entier, le modifier dans votre code, puis réécrire le fichier dans son intégralité.

# Examples

## Redirection

Format:

```
[command] [> | >>] [filename]
```

`>` *enregistre* la sortie de [command] dans [filename].

`>>` *ajoute* la sortie de [commande] dans [nomfichier].

Exemples:

1. `echo Hello World > myfile.txt` enregistre "Hello World" dans myfile.txt
2. `echo your name is %name% >> myfile.txt` ajoute "votre nom est xxxx" dans myfile.txt
3. `dir C:\ > directory.txt` enregistre le répertoire de C: \ à directory.txt

## Echo pour créer des fichiers

Façons de créer un fichier avec la commande echo:

```
ECHO. > example.bat (creates an empty file called "example.bat")

ECHO message > example.bat (creates example.bat containing "message")
ECHO message >> example.bat (adds "message" to a new line in example.bat)
(ECHO message) >> example.bat (same as above, just another way to write it)
```

Si vous souhaitez créer un fichier via la commande `ECHO` , dans un répertoire spécifique de votre ordinateur, vous risquez de rencontrer un problème.

```
ECHO Hello how are you? > C:\Program Files\example.bat

(This will NOT make a file in the folder "Program Files", and might show an error message)
```

Mais alors comment on le fait? Eh bien, c'est en fait extrêmement simple ... Lorsque vous tapez un chemin ou un nom de fichier contenant un espace dans son nom, n'oubliez pas d'utiliser "quotes"

```
ECHO Hello how are you? > "C:\Program Files\example.bat"
(This will create "example.bat" in the folder "Program Files")
```

Mais que faire si vous voulez créer un fichier qui génère un nouveau fichier?

```
ECHO message > file1.bat > example.bat

(example.bat is NOT going to contain "message > file1.bat")
```

```
example.bat will just contain "message"... nothing else
```

Alors comment on fait ça? Eh bien pour cela, nous utilisons le symbole `^` .

```
ECHO message ^> file1.bat > example.bat

(example.bat is going to contain "message > file1.bat")
```

Même chose pour d'autres choses en lot

**L'étape suivante nécessite de connaître les variables et les instructions:**

**cliquez ici pour en savoir plus sur les variables | Cliquez ici pour en savoir plus sur les déclarations if et else**

Variables:

```
SET example="text"
ECHO %example% > file.bat
(This will output "text" to file.bat)
```

Si nous ne voulons pas que "text" soit affiché mais simplement "% example%", alors écrivez:

```
ECHO ^%example^% > file.bat
(This will output "%example%" to file.bat)
```

Instructions IF / ELSE:

```
ELSE statements are written with "pipes" ||

IF ^%example^%=="Hello" ECHO True || ECHO False > file.bat

(example.bat is going to contain "if %example%=="Hello" echo True")
[it ignores everything after the ELSE statement]
```

pour sortir toute la ligne, on fait comme avant.

```
IF ^%example^%=="Hello" ECHO True ^|^| ECHO False > file.bat

This will output:
IF %example%=="Hello" ECHO True || ECHO False
```

Si la variable est égale à "Hello" alors elle dira "True", sinon elle dira "False"

## Enregistrer la sortie de nombreuses commandes

Utilisation de nombreuses commandes `ECHO` pour créer un fichier de commandes:

```
(
  echo echo hi, this is the date today
  echo date /T
```

```
  echo echo created on %DATE%
  echo pause >nul
)>hi.bat
```

L'interpréteur de commandes traite l'ensemble de la section entre parenthèses sous la forme d'une seule commande, puis enregistre toutes les sorties dans `hi.bat` .

`hi.bat` contient maintenant:

```
echo hi, this is the date today
date /T
echo created on [date created]
pause >nul
```

Lire Création de fichiers à l'aide de lots en ligne: https://riptutorial.com/fr/batch-file/topic/7129/creation-de-fichiers-a-l-aide-de-lots

# Chapitre 10: Différences entre lot (Windows) et terminal (Linux)

## Introduction

Lot et bash sont très différents. Les indicateurs de lot sont indiqués par un `/` , alors que les indicateurs bash utilisent un `-` . La capitalisation est importante en bash, mais pas du tout en batch. Les noms de variables par lots peuvent contenir des espaces, mais pas les noms de variables bash. En fin de compte, les deux sont des moyens de manipuler et d'interagir avec la ligne de commande. Sans surprise, il existe une quantité raisonnable de chevauchement entre les capacités des deux systèmes.

## Remarques

- `bitsadmin` est déconseillé en faveur de l'applet de commande PowerShell BITS mais fonctionne toujours à partir de Windows 10 version 1607
- `certutil` sépare les paires de chiffres hexadécimaux avec un espace, donc `md5sum` renvoie un exemple de valeur de `d41d8cd98f00b204e9800998ecf8427e` , alors que `certutil` affiche la même valeur que `d4 1d 8c d9 8f 00 b2 04 e9 80 09 98 ec f8 42 7e`
- `cd` vers un autre lecteur (par exemple, de C: à D :), le drapeau `/d` doit être utilisé
- `del` ne peut pas supprimer les dossiers, utilisez plutôt `rm`
- `grep` est tellement plus puissant que `find` et `findstr` , il n'est pas juste de les comparer; `find` n'a aucune capacité d'expression rationnelle et `findstr` possède des capacités d'expression rationnelle extrêmement limitées ( `[az]{2}` n'est pas une syntaxe valide, mais `[az][az]` est)
- `for` boucles à l'invite de commande Windows ne peut utiliser que des noms de variable à caractère unique; c'est la seule fois que les noms de variables de lot sont sensibles à la casse
- `for` boucles à l'invite de commande, utilisez également la forme de variable `%A` au lieu de `%A%` - `for` boucles dans les scripts de commandes, utilisez la forme de variable `%%A`
- `md` crée automatiquement tous les répertoires parents nécessaires, alors que `mkdir` besoin de l' `-p` pour le faire
- `rem` ne peut pas être utilisé comme caractère de commentaire en ligne à moins qu'il ne soit précédé par un `&`
- `::` ne peut pas être utilisé comme un commentaire en ligne du tout, et ne doit pas non plus être utilisé dans un bloc de code (ensemble de parenthèses)

- Notez que certaines commandes Windows comme `ping` utilisent toujours `-` comme des indicateurs

## Examples

### Commandes par lots et leurs équivalents Bash

| Lot | Frapper | La description |
|---|---|---|
| `command /?` | `man command` | Affiche l'aide pour la *commande* |
| `bitsadmin` | `wget` ou `curl` | Télécharge un fichier distant |
| `certutil -hashfile file_name MD5` | `md5sum file_name` | Obtient la somme de contrôle MD5 de *nom_fichier* |
| `cd` | `pwd` | Affiche le répertoire actuel |
| `cd directory` | `cd directory` | Change le répertoire en cours en celui spécifié |
| `cls` | `clear` | Efface l'écran |
| `copy` | `cp` | Copie un fichier ou des fichiers d'un chemin source vers un chemin cible |
| `date` | `date` | Affiche la date ou la définit en fonction de l'entrée de l'utilisateur |
| `del` | `rm` | Supprime un fichier ou des fichiers |
| `dir` | `ls` | affiche une liste de fichiers et de répertoires dans le répertoire en cours |
| `echo` | `echo` | Affiche le texte à l'écran |
| `exit` | `return` | Quitte un script ou une sous-routine |
| `exit` | `logout` | Ferme l'invite de commande ou le terminal |
| `fc` | `diff` | Compare le contenu de deux fichiers |
| `find "string" file_name` | `grep "string" file_name` | Recherche *nom_fichier* pour *chaîne* |
| `findstr "string" file_name` | `grep "string" file_name` | Recherche *nom_fichier* pour *chaîne* |
| `for /F %A in (fileset*) do something` | `for item in fileset*; do; something; done` | Faites quelque chose pour chaque fichier dans un |

| Lot | Frapper | La description |
|---|---|---|
| | | ensemble de fichiers |
| `for /F %A in ('command') do something` | `` `command` `` | Renvoie la sortie d'une commande |
| `for /L %A in (first,increment,last) do something` | `for item in `seq first increment last`; do; something; done` | Commence par le *début* et compte par *incrément* jusqu'à ce qu'il atteigne le *dernier* |
| `forfiles` | `find` | Recherche des fichiers correspondant à certains critères |
| `if "%variable%"=="value" (` | `if [ "variable"="value" ]; then` | Compare deux valeurs |
| `ipconfig` | `ifconfig` | Affiche les informations IP |
| `md` | `mkdir` | Crée de nouveaux dossiers |
| `mklink` | `ln -s` | Crée un lien symbolique |
| `more` | `more` | Affiche un écran de sortie à la fois |
| `move` | `mv` | Déplace un fichier ou des fichiers d'un chemin source vers un chemin cible |
| `pause` | `read -p "Press any key to continue"` | Suspend l'exécution du script jusqu'à ce que l'utilisateur appuie sur un bouton |
| `popd` | `popd` | Supprime l'entrée supérieure de la pile de répertoires et accède au nouveau répertoire principal |
| `pushd` | `pushd` | Ajoute le répertoire en cours à la pile de répertoires et accède au nouveau répertoire principal |
| `ren` | `mv` | Renomme les fichiers |
| `rem` ou `::` | `#` | Commentaires une ligne de code |
| `rd` | `rmdir` | Supprime les répertoires vides |
| `rd /s` | `rm -rf` | Supprime les répertoires, qu'ils |

| Lot | Frapper | La description |
|---|---|---|
| | | soient vides ou non |
| `set variable=value` | `variable=value` | Définit la valeur de *variable* à *valeur* |
| `set /a variable=equation` | `variable=$((equation))` | Effectue des calculs (batch ne peut utiliser que des entiers de 32 bits) |
| `set /p variable=promptstring` | `read -p "promptstring" variable` | Obtient l'entrée utilisateur et la stocke dans une *variable* |
| `shift` | `shift` | Décale les arguments de 1 (ou n si fourni) |
| `sort` | `sort` | Trie la sortie alphabétiquement par ligne |
| `tasklist` | `ps` | Affiche une liste des processus en cours d'exécution |
| `taskkill /PID processid` | `kill processid` | Tue le processus avec *processid* PID |
| `time /t` | `date` | Affiche l'heure actuelle |
| `type` | `cat` | Affiche le contenu d'un fichier |
| `where` | `which` | Recherche dans le répertoire en cours et le PATH pour un fichier ou une commande |
| `whoami` | `id` | Affiche le nom et le groupe de l'utilisateur actuel |

## Variables de lot et leur équivalent Bash

| Lot | Frapper | La description |
|---|---|---|
| `%variable%` | `$variable` | Une variable régulière |
| `!variable!` | `$variable` | Une variable à l'intérieur d'un bloc de code lorsque `setlocal enabledelayedexpansion` est `setlocal enabledelayedexpansion` |
| `%errorlevel%` ou `ERRORLEVEL` | `$?` | Retourne le statut de la commande précédente: 0 si réussi, 1 (ou autre chose) sinon |

| Lot | Frapper | La description |
| --- | --- | --- |
| `%1` , `%2` , `%3` , etc. | `$1 $2 $3` , etc. | Les paramètres donnés à un script |
| `%*` | `$*` | Tous les paramètres donnés à un script |

Lire Différences entre lot (Windows) et terminal (Linux) en ligne: https://riptutorial.com/fr/batch-file/topic/8362/differences-entre-lot--windows--et-terminal--linux-

# Chapitre 11: Échapper des caractères spéciaux

## Introduction

Dans toutes les versions de `cmd.exe` et `DOS` , certains caractères sont réservés à une utilisation spécifique (par exemple, la redirection de commandes). Ce sujet expliquera comment utiliser les caractères spéciaux sans problèmes.

## Examples

### Échapper à l'aide du caret (^)

La plupart des caractères spéciaux peuvent être échappés en utilisant le curseur ( `^` ). Regardez l'exemple suivant.

```
echo > Hi
echo ^> Hi
```

Cette première commande ne produirait pas `> Hi` car `>` est un caractère spécial, ce qui signifie rediriger la sortie vers un fichier. Dans ce cas, le fichier est nommé "Hi"

Cependant, dans la deuxième commande, `> Hi` sera généré sans aucun problème car le signe d'insertion ( `^` ) indique au `>` cesser de fonctionner comme commande "rediriger la sortie vers le fichier", maintenant `>` est juste un caractère normal.

---

Voici une liste de caractères spéciaux pouvant être échappés (pris et édités à partir de la page de Rob van der Woude)

| Personnage | Résultat échappé | Remarques |
|---|---|---|
| ^ | ^^ | |
| Et | ^ & | |
| < | ^ < | |
| > | ^> | |
| \| | ^ \| | |
| \ | ^ \ | |
| ! | ^^! | Seulement requis lorsque DelayedExpansion est activé |

# S'échapper du caret

Les carets peuvent être empilés jusqu'à l'échappement des autres carets, prenez l'exemple suivant.

| Contribution | Sortie |
|---|---|
| ^ & | Et |
| ^ **^** ^ & | ^ & |
| ^ **^** ^ **^** ^ & | ^^ & |

Note: Les carets en gras sont échappés.

---

# Problème de sécurité

Un peu hors sujet ici, mais c'est très important! Un échappatoire indésirable à la fin du fichier pourrait provoquer une fuite de mémoire!

```
any-invalid-command-you-like-here ^
```

Cette commande fuirait toute la mémoire, **rendant le système complètement inutilisable** ! Voir ici pour plus d'informations.

### Caractères spéciaux FIND et FINDSTR

Dans `find` et `findstr` , certains caractères spéciaux requièrent une certaine prudence.

---

# TROUVER

Il n'y a qu'un seul caractère à échapper - `"` citation. Pour y échapper, ajoutez simplement une autre citation à côté. Alors `"` devient `""` . Assez simple.

---

# FINDSTR

`Findstr` est livré avec plein de personnages pour s'échapper, alors soyez très prudent. En utilisant `\` , nous pouvons échapper à des caractères spéciaux. Voici une liste de caractères spéciaux à échapper

| Personnage | Résultat échappé |
|---|---|
| \ | \\ |
| [ | \ [ |
| ] | \] |
| " | \ " |
| . | \. |
| * | \ * |
| ? | \? |

### Caractères spéciaux FOR / F

## FOR / F

Dans une déclaration `FOR /F` , certains caractères doivent s'échapper, ici une liste (prise et éditée à partir de la page de Rob van der Woude)

| Personnage | Résultat échappé | Remarques |
|---|---|---|
| ' | ^ ' | Seulement nécessaire dans les crochets de `FOR /F` , sauf si `usebackq` est spécifié. |
| ` | ^ ` | Seulement nécessaire dans les crochets de `FOR /F` , quand `usebackq` est spécifié |
| , | ^, | ┐ |
| ; | ^; | ┃ |
| = | ^ = | ┣ Doit être échappé dans les crochets de `FOR /F` , même s'il est entre guillemets |
| ( | ^ ( | ┃ |
| ) | ^) | ┘ |

### Caractères spéciaux supplémentaires

Voici une liste d'autres caractères spéciaux, nécessitant ou pouvant nécessiter une évasion, mais non mentionnés ci-dessus.

| Personnage | Résultat échappé | Remarques |
|---|---|---|
| % | %% | |
| [LF] | ^ [LF] | Cette astuce est citée par Mark Stang dans le groupe de `alt.msdos.batch` . |

## Échapper à travers le pipeline

Quand il y a une expression avec une pipe, la `cmd` commence deux threads des deux côtés de la pipe et l'expression est analysée deux fois (pour chaque côté du pipe) donc les carets doivent être doublés.

Sur le côté gauche:

```
echo ^^^&|more
```

Sur le côté droit:

```
break|echo ^^^&
```

Lire Échapper des caractères spéciaux en ligne: https://riptutorial.com/fr/batch-file/topic/10693/echapper-des-caracteres-speciaux

# Chapitre 12: Écho

## Introduction

`echo` peut être utilisé pour contrôler et produire la sortie.

## Syntaxe

- ECHO [ON | DE]
- Message ECHO
- ECHO (message
- ÉCHO(

## Paramètres

| Paramètre | Détails |
|---|---|
| ON \| DE | Peut être `ON` ou `OFF` (insensible à la casse) |
| message | Toute chaîne (sauf `ON` ou `OFF` lorsqu'elle est utilisée sans `(` ) |

## Remarques

- `echo.` affichera également une chaîne vide. Cependant, cela est plus lent que `echo(` car `echo.` Recherchera un fichier nommé "echo". Ce n'est que si ce fichier n'existe pas que la commande fonctionnera, mais cette vérification le ralentit.
- `echo:` se comportera comme `echo(` sauf si le `message` ressemble à un chemin de fichier, par exemple `echo:foo\..\test.bat` . Dans ce cas, l'interpréteur verra `echo:foo` comme nom de dossier, strip `echo:foo\..\` (car il semble juste pour entrer dans le répertoire `echo:foo` puis le laisser à nouveau) puis exécuter `test.bat` , qui n'est pas le comportement souhaité.

## Examples

### Affichage des messages

Pour afficher "du texte", utilisez la commande:

```
echo Some Text
```

Cela affichera la chaîne `Some Text` suivie d'une nouvelle ligne.

Pour afficher les chaînes `On` et `Off` (insensible à la casse) ou la chaîne vide, utilisez un `(` au lieu d'espace blanc:

```
echo(ON
echo(
echo(off
```

Cela va sortir:

```
ON

off
```

Il est également courant d'utiliser l' `echo.` pour sortir une ligne vide, mais s'il vous plaît voir les remarques pour lesquelles ce n'est pas la meilleure idée.

Pour afficher du texte sans inclure un CR / LF, utilisez la commande suivante:

```
<nul set/p=Some Text
```

Cette commande tentera de définir la variable appelée la chaîne vide à l'entrée de l'utilisateur après une invite. Le fichier `nul` étant redirigé vers la commande avec `<nul` , la commande abandonnera dès qu'elle essaiera de la lire et seule la chaîne d'invite sera laissée. L'utilisateur n'ayant jamais saisi de nouvelle ligne, il n'y a pas de saut de ligne.

### Réglage d'écho

Le paramètre echo détermine si l'écho de commande est activé ou désactivé. C'est ce qu'un exemple de programme ressemble avec la commande en écho (par défaut):

```
C:\Windows\System32>echo Hello, World!
Hello, World!

C:\Windows\System32>where explorer
C:\Windows\System32\explorer.exe

C:\Windows\System32>exit
```

Voici à quoi ça ressemble avec echo **off** :

```
Hello, World!
C:\Windows\System32\explorer.exe
```

# Obtenir et configurer

Pour obtenir (afficher) le paramètre d'écho, utilisez `echo` sans paramètres:

```
> echo
ECHO is on.
```

Pour définir le paramètre d'écho, utilisez l' `echo` avec `on` ou `off` :

```
> echo off

> echo
ECHO is off.

> echo on

> echo
ECHO is on.
```

Notez qu'avec ces exemples, l'invite a été représentée par `>` . Lorsque vous modifiez le paramètre d'écho, l'invite apparaît et disparaît, mais cela crée des exemples peu clairs.

Notez qu'il est possible d'empêcher qu'une commande ne soit répercutée même lorsque l'écho est activé, en plaçant un caractère `@` avant la commande.

## Echo produit tout littéralement

Les devis seront produits tels quels:

```
echo "Some Text"
```

```
"Some Text"
```

Les jetons de commentaire sont ignorés:

```
echo Hello World REM this is not a comment because it is being echoed!
```

```
Hello World REM this is not a comment because it is being echoed!
```

Toutefois, `echo` restituera toujours les variables - voir Utilisation des variables - et les caractères spéciaux restent actifs:

```
echo hello && echo world
```

```
hello
world
```

## Echo à la sortie du fichier

Façons de créer un fichier avec la commande echo:

```
echo. > example.bat (creates an empty file called "example.bat")

echo message > example.bat (creates example.bat containing "message")
echo message >> example.bat (adds "message" to a new line in example.bat)
(echo message) >> example.bat (same as above, just another way to write it)
```

**Sortie sur le chemin**

Un petit problème que vous pourriez rencontrer lors de cette opération:

```
echo Hello how are you? > C:\Users\Ben Tearzz\Desktop\example.bat

(This will NOT make a file on the Desktop, and might show an error message)
```

Mais alors comment on le fait? Eh bien, c'est en fait extrêmement simple ... Lorsque vous tapez un chemin ou un nom de fichier contenant un espace dans son nom, n'oubliez pas d'utiliser "quotes"

```
echo Hello how are you? > "C:\Users\Ben Tearzz\Desktop\example.bat"
(This will make a file on MY Desktop)
```

Mais que faire si vous voulez créer un fichier qui génère un nouveau fichier?

```
echo message > file1.bat > example.bat

(This is NOT going to output:
"message > file1.bat" to example.bat
```

Alors comment on fait ça?

```
echo message ^> file1.bat > example.bat

(This will output:
"message > file1.bat" to example.bat
```

Même chose pour d'autres choses en lot

**Si vous n'avez pas appris quelles sont les variables et les déclarations, alors vous ne comprendrez probablement pas ce qui suit: cliquez ici pour en savoir plus sur les variables | cliquez ici pour en savoir plus sur les déclarations "if"**

Variables:

```
set example="text"
echo %example% > file.bat
(This will output "text" to file.bat)
```

Si nous ne voulons pas que "text" soit affiché mais simplement "% example%", alors écrivez:

```
echo ^%example^% > file.bat
(This will output "%example%" to file.bat)
```

déclarations autres:

```
else = ||
if ^%example^%=="Hello" echo True || echo False > file.bat

(This will output:
if %example%=="Hello" echo True
```

pour sortir toute la ligne, nous écrivons:

```
if ^%example^%=="Hello" echo True ^|^| echo False > file.bat

This will output:
if %example%=="Hello" echo True || echo False
```

Si la variable est égale à "bonjour" alors elle dira "vrai", sinon elle dira "faux"

## @Écho off

`@echo off` empêche l' `@echo off` l'invite et du contenu du fichier de commandes, de sorte que seule la sortie est visible. Le `@` rend également masquée la sortie de la commande `echo off` .

## Faire tourner l'écho sur les parenthèses intérieures

Dans l'exemple suivant, `echo on` prendra effet après la fin du contexte entre parenthèses:

```
@echo off
(
    echo on
    echo ##
)
echo $$
```

Pour "activer" l'écho dans un contexte de parenthèses (y compris les commandes `FOR` et `IF` ), vous pouvez utiliser la macro FOR / f :

```
@echo off
setlocal

:: echo on macro should followed by the command you want to execute with echo turned on
set "echo_on=echo on&for %%. in (.) do"

(
  %echo_on% echo ###
)
```

Lire Écho en ligne: https://riptutorial.com/fr/batch-file/topic/3981/echo

# Chapitre 13: Fichiers aléatoires dans le lot

## Examples

### Nombres aléatoires

En utilisant la variable dynamique `%Random%` , on peut obtenir un entier aléatoire de 0 à 32767. Par exemple:

```
echo %random%
```

Cela retourne évidemment un entier de 0 à 32767. Mais parfois nous voulons qu'il soit dans une plage spécifique, par exemple de 1 à 100.

---

## Génération de nombres aléatoires dans une plage spécifique

La méthode de base pour le faire est répertoriée ci-dessous.

```
set /a result=(%RANDOM%*max/32768)+min
```

où `max` est le plus grand nombre pouvant être généré et `min` est le plus petit nombre pouvant être généré. Notez que vous n'obtiendrez aucun nombre décimal, car vous `set /a` arrondit automatiquement. Pour générer un nombre aléatoire décimal, essayez ceci:

```
set /a whole=(%RANDOM%*max/32768)+min
set /a decimal=(%RANDOM%*max/32768)+min
echo %whole%.%decimal%
```

---

## Génération de nombres aléatoires supérieurs à 32767

Si tu essayes

```
set /a whole=(%RANDOM%*65536/32768)+1
```

vous obtiendrez probablement des nombres aléatoires qui sont impairs.

Pour générer des nombres supérieurs à 32767, voici une meilleure méthode.

```
set /a result=%random:~-1%%random:~-1%%random:~-1%%random:~-1%%random:~-1%%random:~-1%
```

Le code précédent extrait le caractère 1 de chaque `%random%` . Mais cela est fait exprès.

Comme le nombre `random` peut être un nombre à un chiffre, l'extraction du dernier chiffre ne

fonctionnera pas. C'est pourquoi nous n'extrayons que le dernier caractère. Dans ce cas, nous avons 6 `%random:~-1%` , générant le maximum de `999999` , et le minimum à `000000` , vous devrez peut-être ajuster cela pour répondre à vos besoins.

## Pseudo-aléatoire

`cmd.exe` génère la graine en fonction de l'heure à laquelle la section `cmd` a commencé, donc si vous lancez une section multiple au même moment, le résultat risque de ne pas être "aléatoire".

### Alphabets aléatoires

Malheureusement, batch ne dispose pas d'une méthode intégrée pour générer des alphabets, mais en utilisant `%random%` et `for` loop, nous pouvons "générer" des alphabets.

---

Ceci est une idée simple de comment cela fonctionne.

```
set /a result=(%random%*26/32768)+1
for /f "tokens=%result%" %%I in ("A B C D E F G H I J K L M N O P Q R S T U V W X Y Z") do (
    echo %%I
)
```

- Le premier `set /a` instruction génère un nombre aléatoire `N` compris entre 1 et 26
- L'instruction `for /f` sélectionne le `N` ème élément d'une liste de A à Z.
    - Renvoyer le résultat

On peut mettre un total de `31` éléments en 1 `for` boucle, et des éléments pratiquement illimités en utilisant [cette méthode]. ( Lot - pour l'ordre des paramètres de boucle)

### Pseudo-aléatoire Et Aléatoire Aléatoire En Lot

# Distribution pseudo-aléatoire

En accédant à cette réponse Stack Overflow , l'utilisateur CherryDT a souligné ce code:

```
set /a num=%random% %% 100
```

ne donne pas une distribution uniforme.

La variable dynamique interne `%random%` **do** donne une **distribution uniforme** , mais le code ci-dessus ne sera pas un aléatoire uniforme. Ce code génère un nombre aléatoire compris entre 0 et 99, mais le résultat ne sera pas uniforme. 0 ~ 67 se produiront plus de 68 ~ 99 depuis `32767 MOD 100 = 67` .

Pour générer un aléatoire distribué uniforme en utilisant le code ci-dessus, alors `100` doit être modifié. Voici une méthode pour obtenir un nombre qui crée une distribution uniforme.

```
32767 mod (32767 / n)
```

où `n` est un entier, entre 0 et 32767, le résultat peut être décimal et ne pas fonctionner en batch.

---

# Distribution uniforme

```
set /a result=(%RANDOM%*100/32768)+1
```

Cette méthode générera une distribution uniforme. Cela évite d'utiliser `%` , qui ressemble plus à "reste" qu'à "module" dans un script batch. Sans utiliser `%` , le résultat sera uniforme.

Alternativement, voici une méthode inefficace mais uniforme.

```
set /a test=%random%

if %test% geq [yourMinNumber] (
    if %test% leq [yourMaxNumber] (

        rem do something with your random number that is in the range.

   )
)
```

Changez `[yourMinNumber]` et `[yourMaxNumber]` selon vos propres valeurs.

Lire Fichiers aléatoires dans le lot en ligne: https://riptutorial.com/fr/batch-file/topic/10841/fichiers-aleatoires-dans-le-lot

# Chapitre 14: Fichiers batch et hybrides Powershell

## Examples

### Exécuter Powershell avec des fichiers temporaires

Cela a été mentionné dans d'autres sujets hybrides encore et encore. La vieille école, mais la méthode facile pour gérer Powershell est de:

- `echo` au script Powershell dans un script temporaire
- Exécutez le script temporaire
- Supprimez éventuellement le script temporaire

---

Ceci est un exemple de script.

```
@echo off
echo powershell-command>Temp.ps1
echo another line>>Temp.ps1
    rem echo the script into a temporary file

powershell -File Temp.ps1
    rem execute the temporary script

del Temp.ps1
    rem Optionally remove the temporary script
```

La méthode ci-dessus nécessite des tonnes de déclarations d' `echo` si un long script est requis, voici une meilleure méthode suggérée par @Aacini

```
@echo off
setlocal

    rem Get the number of the "<resource>" line
for /F "delims=:" %%a in ('findstr /N "<resource>" "%~F0"') do set "start=%%a"

    rem Skip such number of lines and show the rest of this file
(for /F "usebackq skip=%start% delims=" %%a in ("%~F0") do echo %%a) > Temp.ps1

powershell -File Temp.ps1
del /f /s /q Temp.ps1

goto :EOF

<resource>
PS
Powershell script
```

### Utiliser la commande POWERSHELL pour exécuter la commande Powershell

### à une ligne

A l'aide de la commande `POWERSHELL` , nous pouvons exécuter une commande 1 ligne directement à partir d'un script de commandes, sans fichier temporaire.

---

Voici la syntaxe.

```
powershell.exe -Command <yourPowershellCommandHere>
```

Vous pouvez également inclure d'autres indicateurs, tels que `-Nologo` pour améliorer le résultat réel.

### Powershell / batch hybride sans fichiers temporaires

C'est l'approche proposée par l'utilisateur rojo de stackoverflow, qui peut également gérer les arguments de la ligne de commande:

```
<# : batch portion
@echo off & setlocal

(for %%I in ("%~f0";%*) do @echo(%%~I) | ^
powershell -noprofile "$argv = $input | ?{$_}; iex (${%~f0} | out-string)"

goto :EOF
: end batch / begin powershell #>

"Result:"
$argv | %{ "`$argv[{0}]: $_" -f $i++ }
```

appelé comme ça:

    psbatch.bat arg1 "Ceci est arg2" arg3

produira:

```
Result:
$argv[0]: C:\Users\rojo\Desktop\test.bat
$argv[1]: arg1
$argv[2]: This is arg2
$argv[3]: arg3
```

Lire Fichiers batch et hybrides Powershell en ligne: https://riptutorial.com/fr/batch-file/topic/10711/fichiers-batch-et-hybrides-powershell

# Chapitre 15: Gestion des fichiers dans les fichiers de commandes

## Introduction

Dans cette rubrique, vous allez apprendre à créer, éditer, copier, déplacer et supprimer des fichiers par lots.

## Examples

### Création d'un fichier en lot

Il peut y avoir plusieurs raisons pour lesquelles vous souhaitez créer un fichier texte par lot. Mais quelle que soit la raison, voici comment vous le faites.

Si vous souhaitez remplacer un fichier texte existant, utilisez `>` . Exemple:

```
@echo off
echo info to save > text.txt
```

Mais si vous souhaitez ajouter du texte à un fichier texte déjà utilisé `>>` . Exemple:

```
@echo off
echo info to save >> text.txt
```

Si vous devez enregistrer plusieurs lignes de texte dans un fichier, utilisez `()>text.txt` Exemple:

```
@echo off
(echo username
 echo password)>text.txt
```

### Comment copier des fichiers en batch

Vous souhaiterez peut-être copier des fichiers d'un endroit à un autre. Dans cet exemple, nous allons vous apprendre.

Vous pouvez utiliser la commande `xcopy` . La syntaxe est `xcopy c:\From C:\To`

Exemple:

```
@echo off
xcopy C:\Folder\text.txt C:\User\Username\Desktop
```

Il existe également des commutateurs que vous pouvez utiliser. Si vous souhaitez les afficher, ouvrez l'invite de commandes par le `Start Menu -> Accessories -> Command Prompt` , puis tapez `xcopy`

`/?`

## Déplacement de fichiers

En utilisant la commande move, vous pouvez déplacer les fichiers:

```
@echo off
cd C:\Foo\Bat\Baz
move /Y Meow.bat "Meow Folder" >nul
```

`Meow.bat` représente le fichier à déplacer. Le "Meow Folder" déplace `Meow.bat` dans le `Meow Folder` . `/Y` dit de ne pas demander confirmation. Remplacer cela par `/-Y` rend l'invite du fichier de commandes pour confirmation. Le `>nul` masque la sortie de la commande. S'il n'y avait pas `>nul` , cela produirait:

```
    1 File Moved
```

## Supprimer des fichiers

En utilisant la commande `DEL` (alias for `ERASE` ), vous pouvez supprimer des fichiers.

```
@echo off
del foo.ext
```

Cette commande supprime `foo.ext` du répertoire en cours. On peut également spécifier le chemin et le fichier, tels que:

```
del C:\Foo\Bar\Baz.ext
```

Mais il est toujours idéal de mettre des guillemets ( `"` ) autour des chemins, voir ici pour la raison.

---

Il y a quelques drapeaux disponibles pour `DEL` .

| Drapeau | Fonction |
|---|---|
| `/P` | Invite l'utilisateur avant de supprimer le ou les fichiers |
| `/F` | Supprimer de force le ou les fichiers en lecture seule |
| `/S` | Supprimer le ou les fichiers dans les sous-répertoires |
| `/Q` | Empêche l'invite de l'utilisateur |
| `/A` | Filtre: ne supprime que le fichier attribué spécifique, |
| | en utilisant le caractère `-` signifie **non** attribué comme ce type. |

## Copier des fichiers sans xcopy

Dans cet exemple , l'utilisateur BoeNoe a montré comment utiliser la commande `xcopy` pour copier des fichiers. Il existe également une commande supplémentaire appelée `copy` .

Voici un exemple simple:

```
copy foo.ext bar.ext
```

Cela copie `foo.ext` dans `bar.ext` et crée un `bar.ext` quand il n'existe pas. Nous pouvons également spécifier des chemins vers le fichier, mais il est toujours idéal de placer des guillemets ( `"` ) autour des chemins, voir ici pour la raison.

---

Il y a aussi beaucoup de drapeaux disponibles pour la `copy` , voir `copy /?` ou `help copy` sur une invite de commande pour en savoir plus.

## Modification de la nième ligne d'un fichier

Fichier batch ne vient pas avec une méthode intégrée pour remplacer `n` ième ligne d'un fichier , sauf `replace` et `append` ( `>` et `>>` ). En utilisant `for` boucles, nous pouvons émuler ce type de fonction.

---

```
@echo off
set file=new2.txt

call :replaceLine "%file%" 3 "stringResult"

type "%file%"
pause
exit /b

:replaceLine <fileName> <changeLine> <stringResult>
setlocal enableDelayedExpansion

set /a lineCount=%~2-1

for /f %%G in (%~1) do (
    if !lineCount! equ 0 pause & goto :changeLine
    echo %%G>>temp.txt
    set /a lineCount-=1
)

:changeLine
echo %~3>>temp.txt

for /f "skip=%~2" %%G in (%~1) do (
    echo %%G>>temp.txt
)

type temp.txt>%~1
del /f /q temp.txt

endlocal
exit /b
```

- Le script principal appelle la fonction `replaceLine` , avec le nom de fichier / la ligne à modifier / et la chaîne à remplacer.
- Fonction reçoit l'entrée
    - Il parcourt toutes les lignes et les `echo` à un fichier temporaire avant la ligne de remplacement
    - Il fait `echo` la ligne de remplacement du fichier
    - Il continue à sortir au reste du fichier
    - Il copie le fichier temporaire dans le fichier d'origine
    - Et supprime le fichier temporaire.
- Le script principal récupère le contrôle et `type` le résultat.

Lire Gestion des fichiers dans les fichiers de commandes en ligne: https://riptutorial.com/fr/batch-file/topic/9253/gestion-des-fichiers-dans-les-fichiers-de-commandes

# Chapitre 16: Hybrides Batch et JScript

## Introduction

`JScript` est en fait le surensemble de `Javascript` (sa version 1.8.1 - de sorte que certaines nouvelles fonctionnalités ne sont pas disponibles), et ils peuvent être intégrés dans un `batch` de script pour l' extension `batch` fonctions de script. Habituellement, les techniques d'intégration utilisent les directives JScript (ne faisant pas partie du standard officiel Javascript) afin de séparer le code batch et le code JScript. JScript vous permet de travailler avec des objets Com / ActiveX, ainsi qu'avec des objets WMI en plus du Javascript standard.

## Examples

### JScript intégré dans un fichier batch

Cet exemple suivant est créé par l'utilisateur Michael Dillon à partir de cette réponse .

---

Considérons le script suivant:

```
@set @junk=1 /*
@echo off
cscript //nologo //E:jscript %0 %*
goto :eof
*/

//JScript aka Javascript here
```

Cet extrait de script fait:

- Exécutez la commande `cscript` qui s'appelle avec tous les arguments fournis.
- Comme la partie après `@set @junk=1` est commenté ( `/*` et `*/` sont des commentaires JScript valides),
- JScript les ignorera.
- **Remarque: Nous avons besoin de la partie `@set @junk=1` car le fichier de commandes ne reconnaît pas `/*` comme une commande, mais une instruction `set` sera une solution de contournement. JScript reconnaîtra `/*` comme un commentaire, donc l'autre fichier de `batch` ne sera pas exécuté par le moteur JScript.**

---

Vous pouvez ajouter votre JScript après `*/` et commencer à étendre votre script de fichier batch!

### Exécuter JScript avec des fichiers temporaires

Comme mentionné ici , la méthode ancienne pour exécuter un autre script consiste à utiliser des fichiers temporaires. Simple `echo` dans un fichier, puis exécutez-le (et supprimez-le éventuellement).

---

Voici le concept de base:

```
@echo off
echo //A JS Comment > TempJS.js
echo //Add your code>>TempJS.js

cscript //nologo //e:cscript.exe TempJS.js

del /f /s /q TempJS.js
```

Mais cela nécessiterait beaucoup d'instructions `echo` pour créer un JScript relativement grand. Voici une meilleure méthode par Aacini.

```
@echo off
setlocal

rem Get the number of the "<resource>" line
for /F "delims=:" %%a in ('findstr /N "<resource>" "%~F0"') do set "start=%%a"

rem Skip such number of lines and show the rest of this file
(for /F "usebackq skip=%start% delims=" %%a in ("%~F0") do echo %%a) > TempJS.js

cscript //nologo //e:cscript.txt TempJS.js
del /f /s /q TempJS.js

goto :EOF

<resource>
JScript
JScript
JScript
```

Lire Hybrides Batch et JSCript en ligne: https://riptutorial.com/fr/batch-file/topic/10578/hybrides-batch-et-jscript

# Chapitre 17: Hybrides batch et VBS

## Introduction

`Batch` sont capables de fonctionner avec les fonctionnalités `VBS` , augmentant encore leur fiabilité. Par exemple, `VBS` peut gérer les décimales, les espaces et certaines autres opérations avancées qui ne peuvent pas être effectuées par `batch` . Est également capable de travailler avec des objets WMI et ActiveX.

## Examples

### Exécuter VBS avec des fichiers temporaires

L'ancienne méthode pour exécuter un autre script à partir d'un `batch` consiste à `echo` au script dans un autre emplacement, puis à l'exécuter.

Cette méthode peut être représentée comme ceci:

```
@echo off
rem VBS below
    echo your vbs > TempVBS.vbs
    echo other vbs>>TempVBS.vbs
rem End of VBS

cscript //nologo TempVBS.vbs
del /f /s /q TempVBS.vbs
```

La méthode ci-dessus nécessiterait beaucoup d' `echo (vbs) >> TempVBS.vbs` , alors voici un moyen de le raccourcir. *(code par Aacini)*

```
@echo off
setlocal

rem Get the number of the "<resource>" line
for /F "delims=:" %%a in ('findstr /N "<resource>" "%~F0"') do set "start=%%a"

rem Skip such number of lines and show the rest of this file
(for /F "usebackq skip=%start% delims=" %%a in ("%~F0") do echo %%a) > Program.vbs

cscript //nologo Program.vbs
del /f /s /q Program.vbs
exit /b

<resource>
your vbs
another line of vbs
```

La dernière méthode consiste à utiliser des `streams` . Un fichier peut avoir quelques flux. Et chaque flux peut contenir des informations différentes.

```
@echo off

    echo vbs >%0:stream1
    rem This command redirect "vbs" into the stream1 of this script, then we can call it later

cscript %0:stream1 //nologo
   rem if you like, you can clear the stream1 of this file by:
   type nul>%0:stream1
```

## Intégrer du code vbscript dans un fichier de commandes sans utiliser de fichiers temporaires

Voici un exemple avec la technique (hack) inventée par l'utilisateur Liviu des forums dostips :

```
@echo off
echo Printed by CMD.EXE
cscript //nologo "%~f0?.wsf" //job:JS //job:VBS

exit /b %errorlevel%

----END OF BATCH CODE---
<package>
  <job id="JS">
    <script language="VBScript">

        WScript.Echo("Printed by VBScript"):

    </script>
  </job>
  <job id="VBS">
    <script language="JScript">

        WScript.Echo("Printed by JScript");

    </script>
  </job>
</package>
```

Comme exécuter `wsf` fichier `wsf` avec Windows script host est sensible aux extensions, vous pouvez exécuter un fichier avec n'importe quelle extension en ajoutant `?.wsf` à la fin du fichier (qui est le noyau du hack). Alors que l'exemple de Liviu est probablement plus robuste, le code ci-dessus est une version plus simplifiée. Comme wsh ne se soucie guère des choses en dehors du nœud `<package>` vous n'êtes pas obligé de tout mettre dans les commentaires XML. Bien que ce soit prudent avec les symboles de redirection ( `<` et `>` )

Lire Hybrides batch et VBS en ligne: https://riptutorial.com/fr/batch-file/topic/10061/hybrides-batch-et-vbs

# Chapitre 18: Les fonctions

## Remarques

Vous pouvez ajouter des variables de démarrage à la fonction en ajoutant `<parameter>` à son libellé. Ces variables de démarrage sont accessibles avec `%n` où n est le numéro de la variable de départ ( `%1` pour le premier, `%2` pour le second. Cette méthode `%n` fonctionne pour% 1 -% 9. Pour le paramètre 10 - 255, vous aurez besoin de utiliser la commande Shift ).
Par exemple:

```
:function <var1> <var2>
```

Une fois que vous utilisez l' `call :function param1 param2` , `param1` est accessible avec `%1` et `param2` avec `%2` .
Remarque: le `<parameter>` n'est pas strictement nécessaire, mais il aide à la lisibilité.

Une astuce utile qui est utile lorsque beaucoup de variables volent est d'utiliser `setlocal` et `endlocal` en tandem avec `%n` . `setlocal` et `endlocal` font essentiellement de la fonction une instance distincte de l'invite de commande, les variables qu'elle `endlocal` que `endlocal` dans le cadre.

Si vous utilisez `setlocal` et `endlocal` et que vous retournez des valeurs globales, utilisez ceci.

```
endlocal & set var=variable
```

Cela définit la valeur globale `var` sur `variable` . Vous pouvez les enchaîner pour plusieurs variables.

```
endlocal & set var=variable & set var2=variable number 2
```

Cela définit la variable globale `var` à `variable` et la valeur globale `var2` à la `variable number 2` .
Puisque le code dans les blocs de code est également exécuté simultanément, vous pouvez également le faire.

```
if "%var%"=="" (
    endlocal
    set %~2=10
)
```

Mais vous **ne pouvez pas le** faire.

```
if "%var%"=="" (
    set %~2=10
    endlocal
)
```

## Examples

## Fonction simple

```
call :FunctionX
rem More code...

:FunctionX
rem Some code here.
goto :eof
```

C'est une fonction très simple. Les fonctions sont des commandes dans le programme qui exécutent plusieurs commandes à la fois. Les fonctions sont créées en créant une étiquette et en y insérant du code, et une fois cela fait, vous ajoutez un `goto :eof` ou `exit /b <ErrorlevelYou'dLike>` qui renvoie à l'endroit où il a été appelé. Les fonctions sont appelées avec l' `call :functionname adparams` .

## Fonction avec paramètres

```
call :tohex 14 result
rem More code...

:tohex <innum> <outvar>
set dec=%1
set outvar=%~2
rem %n and %~n are functionally identical, but %~n is slightly safer.
goto :eof
```

Cela prend les paramètres supplémentaires de l' `call` comme si la fonction était un fichier de lot distinct.
Remarque: le `<parameter>` n'est pas nécessaire, mais il aide à la lisibilité.

## Fonction utilisant setlocal et endlocal

```
set var1=123456789
set var2=abcdef
call :specialvars
echo %var1%, %var2%
rem More code...

:specialvars
setlocal
set var1=987654321
set var2=fedcba
endlocal
goto :eof
```

Lorsque dans la section `setlocal` , section `endlocal` , les variables sont séparées des variables de l'appelant, pourquoi% var1% et% var2% n'ont pas été modifiés.

## Les combiner tous

```
set importantvar=importantstuff
```

```
call :stuff 123 var1
rem More code...

:stuff <arg1> <arg2>
setlocal
set importantvar=%~1
echo Writing some stuff into %~2!
endlocal
set %~2=some stuff
setlocal
set importantvar=junk
endlocal
goto :eof
```

Cela utilise la fonction de base, `setlocal` et `endlocal` et des arguments pour créer une petite fonction étrange.

## Fonctions anonymes dans les fichiers de commandes

La technique des fonctions anonymes utilise le fait que la commande `CALL` utilise en interne `GOTO` lorsqu'une sous-routine est appelée et qu'elle utilise de manière abusive l'impression des messages d'aide avec une double extension variable :

```
@echo off
setlocal
set "anonymous=/?"

call :%%anonymous%% a b c 3>&1 >nul

if "%0" == ":%anonymous%" (
  echo(
  echo Anonymous call:
  echo %%1=%1 %%2=%2 %%3=%3
  exit /b 0
)>&3
```

Vous pouvez appeler une fonction anonyme uniquement si elle est définie après l'appel `CALL` (ou après avoir terminé le contexte des parenthèses si l' `CALL` est exécuté entre parenthèses). Il ne peut pas être appelé à partir d'un script externe , mais est un appel de fonction plus lent que la normale.

## Fonctions d'appel d'un autre fichier de commandes

Avons le fichier suivant appelé **library.cmd** :

```
@echo off

echo -/-/- Batch Functions Library -/-/-

:function1
    echo argument1 - %1
    goto :eof
```

Pour n'exécuter que le **: function1** sans le code du reste du fichier, vous devez mettre une

étiquette **: function1** dans l'appelant et l'utiliser comme ceci:

```
@echo off

call :function1 ###
exit /b %errorlevel%

:function1
    library.bat %*
```

le résultat sera (le code en dehors de la fonction dans `library.cmd` n'est pas exécuté):

    argument1 - ###

Pour plus d'informations, cochez cette case.

Lire Les fonctions en ligne: https://riptutorial.com/fr/batch-file/topic/7646/les-fonctions

# Chapitre 19: Les meilleures pratiques

## Introduction

Ce sujet se concentrera sur les choses que l'on devrait *(pas obligatoire)* faire dans un fichier de commandes. L'utilisation de ces "meilleures pratiques" peut améliorer l'effet et la fonction d'un fichier de commandes.

## Examples

### Citations

La plupart des scripts de lot en ligne comportent de nombreux problèmes de devis.

---

# Exemples et solutions

## Exemple A

```
if %var%==abc echo Test
```

Ce code fonctionne - lorsque le contenu de `%var%` ne contient pas d'espace ou d'autres caractères spéciaux. Supposons maintenant que `%var%` contient 1 espace. Maintenant, `cmd.exe` voit:

```
if  ==abc echo Test
```

Cela provoquerait un échec car `cmd.exe` ne comprend pas cette syntaxe.

## Solution A

```
if "%var%"=="abc" echo Test
```

En utilisant des guillemets, `cmd.exe` considère l'intégralité de `%var%` (y compris l'espace et les caractères spéciaux) comme une seule chaîne normale. Pourtant, ce n'est pas la méthode de comparaison la plus sûre. Le plus sûr utilise `echo` , `pipe` et `findstr` .

---

## Exemple b

```
cd C:\User\Spaced Name\Spaced FileName.txt
```

`cd`

ne changera que le répertoire en `C:\User\Spaced` , car `cd` accepte uniquement un argument de chemin.

## Solution B

Simplement en ajoutant des guillemets sur le chemin, le problème serait résolu.

```
cd "C:\User\Spaced Name\Spaced FileName.txt"
```

---

Il y a aussi quelques exemples qui fonctionnent mieux en utilisant des guillemets, comme `set /a` , etc.

### Code Spaghetti

Le code spaghetti signifie un extrait de code qui utilise de nombreuses structures, souvent déroutantes. Tels que `GOTO` , exceptions et code incohérent.

---

# Exemples et solutions

## Exemple A

```
@echo off
set /a counter=0

:Loop
set /a counter=%counter% + 1
echo %counter%

if %counter% equ 10 goto :exit
goto :Loop

:exit
```

Ce programme est livré avec beaucoup de sauts, ce qui nous rend plus difficile de savoir exactement ce que fait le script.

## Solution A

```
@echo off
for /l %%G in (0,1,10) echo %%G
```

En utilisant moins de `GOTO` , nous avons considérablement réduit la quantité de code et nous pouvons nous concentrer sur le code réel.

---

## Exemple b

Considérez les déclarations suivantes.

```
:endGame
if %player1Score% gtr %player2Score% goto :player1wins
if %player1Score% lss %player2Score% goto :player2wins
goto :tie

:player1wins
echo player 1 wins
goto :eof

:player2wins
echo player 2 wins
goto :eof

:tie
echo tie
goto :eof
```

Cet extrait nécessite beaucoup d' `goto` et peut être source de confusion pour le débogage. Pour simplifier ces instructions, nous pouvons utiliser la commande d' `call` . Voici le script ci-dessus dans une meilleure condition.

```
:endGame
if %player1Score% gtr %player2Score% call :message player 1 wins
if %player1Score% lss %player2Score% call :message player 2 wins
if %player1Score% equ %player2Score% call :message tie

goto :eof

:message
echo %*
goto :eof
```

Les deux scripts produisent exactement le même résultat, mais le nouveau script est beaucoup plus court et plus clair.

Lire Les meilleures pratiques en ligne: https://riptutorial.com/fr/batch-file/topic/10746/les-meilleures-pratiques

# Chapitre 20: Macros de fichiers par lots

## Introduction

Dans une invite de commande, vous pouvez utiliser DOSKEY pour créer des macros. Dans un fichier de commandes, vous pouvez définir une variable pouvant être appelée comme un morceau de code et même lui transmettre des arguments.

## Examples

### Macro de base

En utilisant `DOSKEY` , nous pouvons créer des macros pour simplifier la saisie de nombreuses commandes dans l'invite de commande. Regardez l'exemple suivant.

```
DOSKEY macro=echo Hello World
```

Maintenant, si vous tapez `macro` dans l'invite de commande, il renverrait `Hello World` .

### commentaires

Malheureusement, la macro `DOSKEY` ne supporte pas les commentaires, mais il existe une solution de contournement.

```
;= Comment
;= Comment
;= Remember to end your comment with ;=
;=
```

### $ Caractères Usages

Il y a 3 utilisations du caractère `$` dans une macro `DOSKEY` .

---

# Séparateur de commandes

`$T` est l'équivalent de `&` dans un script de commandes. On peut joindre des commandes ensemble comme ça.

```
DOSKEY test=echo hello $T echo world
```

---

# Arguments de ligne de commande

Comme `bash` (pas par `batch` ), nous utilisons `$` pour indiquer un argument de ligne de commande.

`$1` fait référence au premier argument de ligne de commande

`$2` fait référence au deuxième argument de la ligne de commande, etc.

`$*` fait référence à tous les arguments de la ligne de commande

## Macros en script batch

`DOSKEY` macros `DOSKEY` ne fonctionnent pas dans un script batch. Cependant, nous pouvons utiliser une petite solution de contournement.

```
set DOSKEYMacro=echo Hello World
%DOSKEYMacro%
```

Ce script peut simuler la fonction macro. On peut aussi utiliser des esperluettes ( `&` ) pour joindre des commandes, comme `$T` dans `DOSKEY` .

Si vous voulez une "macro" relativement grande, vous pouvez essayer une fonction simple ou consulter d'autres rubriques de fonction ici .

Lire Macros de fichiers par lots en ligne: https://riptutorial.com/fr/batch-file/topic/10791/macros-de-fichiers-par-lots

# Chapitre 21: Pile de répertoire

## Syntaxe

- PUSHD [chemin]
- POPD

## Paramètres

| Paramètre | Détails |
|---|---|
| chemin | Le répertoire pour naviguer |

## Remarques

- Utiliser `pushd` paramètres ne permet d'imprimer la pile.
- La commande `popd` remplace la valeur actuelle du répertoire actuel.

## Examples

### Supprimer des fichiers texte

L'exemple suivant montre comment vous pouvez utiliser la commande `pushd` et la commande `popd` dans un programme de traitement par lots pour modifier le répertoire en cours à partir de celui dans lequel le programme de traitement par lots a été exécuté, puis le rétablir:

```
@echo off
rem This batch file deletes all .txt files in a specified directory
pushd %1
del *.txt
popd
cls
echo All text files deleted in the %1 directory
```

Source: https://technet.microsoft.com/en-us/library/cc771180%28v=ws.11%29.aspx

### Imprimer répertoire pile

Pour imprimer la pile de répertoires, utilisez la commande `pushd` sans aucun paramètre:

```
@echo off

cd C:\example\
pushd one
```

```
pushd ..\two
pushd ..\..


pushd
echo Current Directory: %cd%

echo:
popd
pushd three

pushd
echo Current Directory: %cd%
```

Sortie:

```
C:\example\two
C:\example\one
C:\example
Current Directory: C:\

C:\example\two
C:\example\one
C:\example
Current Directory: C:\example\two\three
```

Lire Pile de répertoire en ligne: https://riptutorial.com/fr/batch-file/topic/4288/pile-de-repertoire

# Chapitre 22: Pour les boucles dans les fichiers batch

## Syntaxe

- pour / l %% p dans (startNumber, increment, endNumber)
- pour / f %% p dans (nomfichier) commande
- for / f %% p in ("textStrings") ne commande
- pour / f %% p in ('commande')
- pour le lecteur / r: \ path %% p in (set) commande
- pour / d %% p dans (répertoire)

## Remarques

La commande `for` accepte les options lorsque l'option `/f` est utilisée. Voici une liste d'options pouvant être utilisées:

- `delims=x` Caractère de `delims=x` pour séparer les jetons
- `skip=n` Nombre de lignes à ignorer au début du fichier et des chaînes de texte
- `eol=;` Caractère au début de chaque ligne pour indiquer un commentaire
- `tokens=n` Eléments numérotés à lire de chaque ligne ou chaîne à traiter
- `usebackq` Utilisez un autre style de citation:

  Utilisez des guillemets doubles pour les noms de fichiers longs dans "fichiers"

  Utiliser des guillemets simples pour les 'textStrings'

  Utilisez des guillemets pour `command`

## Examples

### Faire une boucle sur chaque ligne d'un ensemble de fichiers

Ce qui suit fera écho à chaque ligne du fichier `C:\scripts\testFile.txt` . Les lignes vierges ne seront pas traitées.

```
for /F "tokens=*" %%A in (C:\scripts\testFile.txt) do (
  echo %%A
  rem do other stuff here
  )
```

Un exemple plus avancé montre comment les données de l'ensemble de fichiers restreints peuvent être utilisées pour rediriger l'exécution en mode batch, tout en sauvegardant le contenu recherché dans un fichier:

```
@echo off
setlocal enabledelayedexpansion

for /f %%i in ('dir "%temp%\test*.log" /o:-d /t:w /b') do (
    set "last=%temp%\%%i"
    type !last! | find /n /i "Completed" >nul 2>&1 >> %temp%\Completed.log ^
     && (echo Found in log %%i & goto :end) || (echo Not found in log %%i & set "result=1"))

:: add user tasks code here
if defined result echo Performing user tasks...

:end
echo All tasks completed
exit /b
```

Notez combien de temps les chaînes de commande sont divisées en plusieurs lignes de code et que les groupes de commandes sont séparés par des parenthèses.

## Visite récursive des répertoires dans une arborescence de répertoires

`for /r` commande `for /r` peut être utilisée pour visiter récursivement tous les répertoires d'une arborescence de répertoires et exécuter une commande.

```
@echo off
rem start at the top of the tree to visit and loop though each directory
for /r %%a in (.) do (
  rem enter the directory
  pushd %%a
  echo In directory:
  cd
  rem leave the directory
  popd
  )
```

Remarques:

- for / r - Boucle à travers les fichiers (sous-dossiers Recurse).
- pushd - Modifie le répertoire / dossier en cours et stocke le dossier / chemin précédent à utiliser par la commande POPD.
- popd - Retourne le répertoire dans le chemin / dossier le plus récemment stocké par la commande PUSHD.

## Renommer tous les fichiers du répertoire en cours

Ce qui suit utilise une variable avec une boucle `for` pour renommer un groupe de fichiers.

```
SetLocal EnableDelayedExpansion

for %%j in (*.*) do (
```

```
set filename=%%~nj
set filename=!filename:old=new!
set filename=Prefix !filename!
set filename=!filename! Suffix
ren "%%j" "!filename!%%~xj"
)
```

En définissant le nom de la variable `%%j` et en l'associant à tous les fichiers actuels `(*.*)` , Nous pouvons utiliser la variable dans une boucle `for` pour représenter chaque fichier du répertoire en cours.

Chaque itération (ou passe) de la boucle traite ainsi un fichier différent du groupe défini (qui pourrait également être n'importe quel groupe, par exemple `*.jpg` ou `*.txt` ).

Dans le premier exemple, nous substituons le texte: la chaîne de texte "old" est remplacée par la chaîne de texte "new" (si, mais seulement si, le texte "old" est présent dans le nom du fichier).

Dans le second exemple, nous ajoutons du texte: le texte "Préfixe" est ajouté au début du nom du fichier. Ce n'est pas une substitution. Cette modification sera appliquée à tous les fichiers du groupe.

Dans le troisième exemple, nous ajoutons à nouveau du texte: le texte "Suffixe" est ajouté à la fin du nom du fichier. Encore une fois, ce n'est pas une substitution. Cette modification sera appliquée à tous les fichiers du groupe.

La dernière ligne gère réellement le changement de nom.

### Itération

```
for /L %%A in (1,2,40) do echo %%A
```

Cette ligne sera itérative de 1 à 39, augmentant de 2 à chaque fois.

Le premier paramètre, `1` , est le numéro de départ.

Le deuxième paramètre, `2` , est l'incrément.

Le troisième paramètre, `40` , est le maximum.

Lire Pour les boucles dans les fichiers batch en ligne: https://riptutorial.com/fr/batch-file/topic/3695/pour-les-boucles-dans-les-fichiers-batch

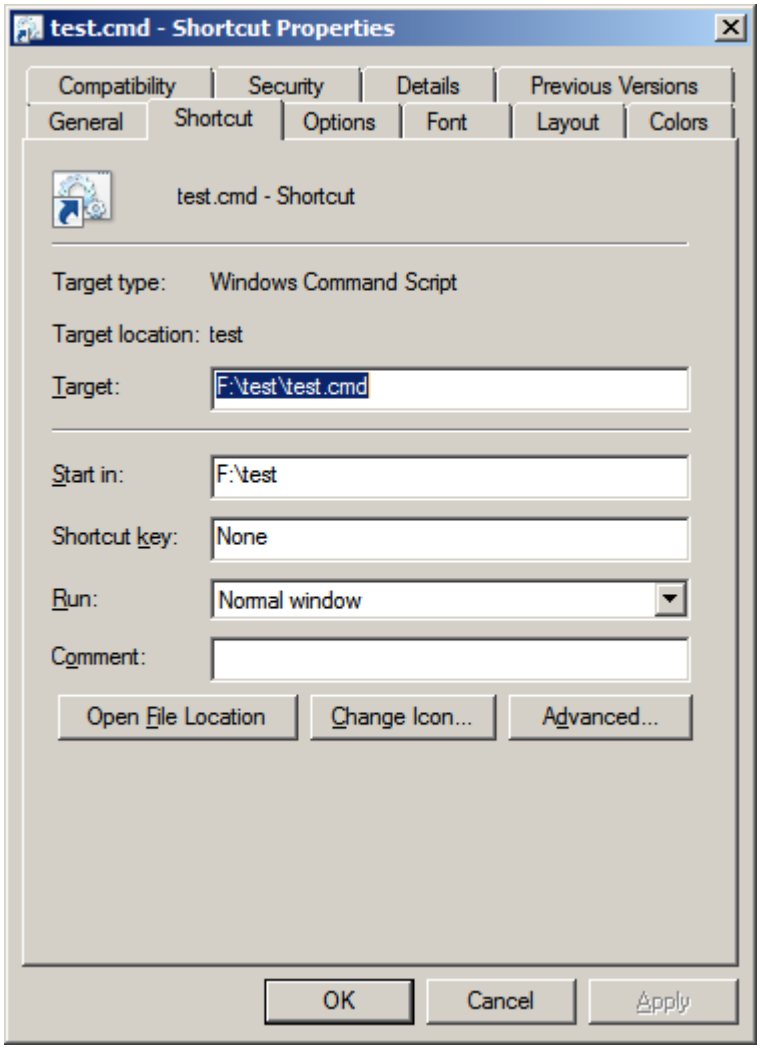
# Chapitre 23: Privilèges élevés dans les fichiers batch

## Examples

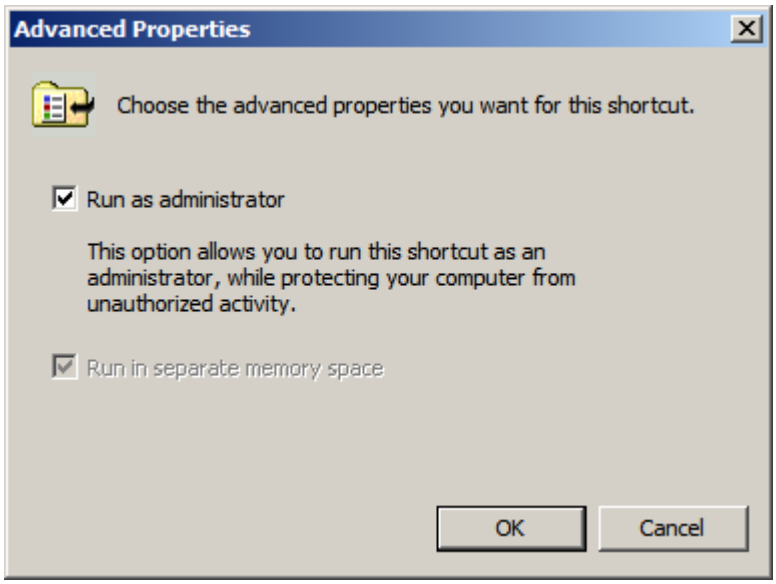
### Demande d'élévation de privilèges dans un raccourci

De nombreuses tâches nécessitent des privilèges élevés. Vous pouvez élever les privilèges d'utilisateur au niveau Administrateur pour votre exécution par lots en utilisant un raccourci:

1. Appuyez sur `alt` + et le fichier de commandes dans un dossier sélectionné pour créer un raccourci.
2. Faites un clic droit et sélectionnez "Propriétés".
3. Sélectionnez l'onglet "Raccourci".
4. Cliquez sur "Avancé".




5. Activer "Exécuter en tant qu'administrateur".




6. Cliquez deux fois sur "OK".

## Demander des privilèges élevés à l'exécution

Le fichier de commandes suivant affichera une invite UAC vous permettant d'accepter les privilèges Administrateur élevés pour la session par lots. Ajoutez le code de vos tâches à `:usercode` section `:usercode` du lot afin qu'elles s'exécutent avec des privilèges élevés.

```
@echo off
setlocal EnableDelayedExpansion
:: test and acquire admin rights
cd /d %~dp0 & echo/
if not "%1"=="UAC" (
    >nul 2>&1 net file && echo Got admin rights || (echo No admin rights & ^
MSHTA "javascript: var shell = new ActiveXObject('shell.application');
shell.ShellExecute("%~snx0", 'UAC', '', 'runas', 1);close();"))
:: re-test admin rights
echo/ & >nul 2>&1 net file && (echo Got admin rights & echo/) || (echo No admin rights.
Exiting... & goto :end)

:usercode
:: add your code here
echo Performing admin tasks
echo Hello >C:\test.txt

:end
timeout /t 5 >nul
exit /b
```

## Demande de privilèges d'exécution sans invite UAC

Comme exemple précédent, ce script demande une élévation si nécessaire. Nous demandons à l'utilisateur des informations d'identification évitant l'invite UAC.

```
@echo off

cls & set "user=" & set "pass="
```

```
:: check if we have administrative access
:: ---------------------------------------------------------------------
whoami /groups | find "S-1-5-32-544">NUL 2>&1 && goto:gotAdmin


echo/
echo/ Testing administrative privileges
echo/
echo/ Please enter administrative credentials
echo/

:: get user
:: ---------------------------------------------------------------------
set/P user="*     User:: "
if "%user%" EQU "" exit/B 1
:: get password
:: ---------------------------------------------------------------------
<NUL set/p=* password& call:getPass pass || exit/B 1
if "%pass%" EQU "" exit/B 1

:: check if credential has administrative privileges
:: this call can be removed
:: ---------------------------------------------------------------------
call:askIsAdmin "%user%", "%pass%" || goto:notAdmin

:: run elevated without UAC prompt
:: with the previous call enabled, we are sure credentials are right
:: without it, this will fail if credentials are invalid
:: ---------------------------------------------------------------------
call:elevateScript "%user%", "%pass%"

exit/B 0

:gotAdmin
echo(
echo( You have administrative rights.
echo(
timeout /T 7
exit/B 0

:notAdmin
echo(
echo( Invalid credential or non administrative account privileges
echo(
timeout /T 7
exit/B 1

:: invoke powershell to get the password
:: ---------------------------------------------------------------------
:getPass
SetLocal
set "psCmd=powershell -Command "$pwd = read-host ':' -AsSecureString;
$BSTR=[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($pwd);
[System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)""
for /F "usebackq delims=" %%# in (`%psCmd%`) do set "pwd=%%#"
if "%pwd%" EQU "" EndLocal & exit/B 1
EndLocal & set "%1=%pwd%"
doskey /listsize=0 >NUL 2>&1 & doskey /listsize=50 >NUL 2>&1        & rem empty keyboard
buffer
exit/B 0
```

```
:: check if credential has administrative privileges
:: -----------------------------------------------------------------------
:askIsAdmin
SetLocal & set "u=%~1" & set "p=%~2" & set/A ret=1
set "psCmd=powershell -Command "$p='%p%'^|convertto-securestring -asplaintext -force;$c=new-
object -typename system.management.automation.pscredential('%u%',$p^);start-process
'powershell' '-Command "write-host ([Security.Principal.WindowsPrincipal]
[Security.Principal.WindowsIdentity]::GetCurrent(^)^).IsInRole([Security.Principal.WindowsBuiltInRole]
-credential $c -passthru -wait;""
for /F "usebackq delims=" %%# in (`%psCmd%`) do @set "result=%%#"
echo %result% | find /I "true">NUL 2>&1 && set/A ret=0
EndLocal & exit/B %ret%
exit/B 1

:: run elevated without UAC prompt
:: -----------------------------------------------------------------------
:elevateScript
SetLocal & set "u=%~1" & set "p=%~2"
set "_vbs_file_=%TEMP%\runadmin.vbs"
echo set oWS ^= CreateObject^("wScript.Shell"^)>"%_vbs_file_%"
echo strcmd="C:\Windows\system32\runas.exe /user:%COMPUTERNAME%\%u% " +
"""%~dpnx0""">>"%_vbs_file_%"
echo oWS.run strcmd, 2, false>>"%_vbs_file_%"
echo wScript.Sleep 100>>%_vbs_file_%
echo oWS.SendKeys "%p%{ENTER}">>%_vbs_file_%
if exist "%TEMP%\runadmin.vbs" (set "_spawn_=%TEMP%\runadmin.vbs") else (set
"_spawn_=runadmin.vbs")
ping 1.1.1.1 -n 1 -w 50 >NUL
start /B /WAIT cmd /C "cls & "%_spawn_%" & del /F /Q "%_spawn_%" 2>NUL"
EndLocal & set "pass="
exit/B 0
```

Lire Privilèges élevés dans les fichiers batch en ligne: https://riptutorial.com/fr/batch-file/topic/4898/privileges-eleves-dans-les-fichiers-batch

# Chapitre 24: Rechercher des chaînes dans un lot

## Examples

### Recherche de chaînes de base

La commande FIND peut analyser des fichiers volumineux ligne par ligne pour trouver une chaîne donnée. Il ne prend pas en charge les caractères génériques dans la chaîne de recherche.

```
find /i "Completed" "%userprofile%\Downloads\*.log" >> %targetdir%\tested.log

TYPE scan2.txt | FIND "Failed" /c && echo Scan failed || echo Scan Succeeded
```

La commande FINDSTR a plus de fonctionnalités et prend en charge la recherche par expressions régulières (REGEX) avec des caractères génériques dans la chaîne de recherche.

```
FINDSTR /L /C:"Completed" Results.txt

echo %%G | findstr /r /b /c:"[ ]*staff.*" >nul && echo Found!
```

Voir les sources d'aide FIND et FINDSTR pour plus d'informations.

### Utilisation des résultats de recherche

Le script suivant montre une technique de *fichiers fractionnés* plus avancée, où la fonction FORF parcourt une liste de fichiers dans un répertoire, et chaque contenu de fichier est redirigé vers FINDSTR qui recherche une chaîne contenant une sous-chaîne `var` précédée d'un nombre indéfini d'espaces. texte. Une fois trouvé, le fichier recherché est remplacé par un nouveau fichier contenant uniquement la partie texte au-dessus de la chaîne de recherche.

```
@echo off
setlocal enabledelayedexpansion
pushd "%temp%\Test"
for %%G in ("*.txt") do (set "break="
    (for /f "tokens=*" %%H in (%%~G) do (
        if not defined break (
            echo %%H | findstr /r /b /c:"[ ]*var.*" >nul && set break=TRUE || echo %%H )
    )) >> %%~nG_mod.txt
    del %%~G & ren %%~nG_mod.txt %%G )
popd
exit /b
```

Notez que la définition de `break=TRUE` permet de quitter la boucle FOR à partir du fichier recherché, une fois que la première occurrence de la chaîne de recherche est trouvée.

Lire Rechercher des chaînes dans un lot en ligne: https://riptutorial.com/fr/batch-

file/topic/5476/rechercher-des-chaines-dans-un-lot

# Chapitre 25: Redirection d'entrée et de sortie

## Syntaxe

- [commande] [[> | >> | <| 2> | 2 >>] fichier]
- [[> | >> | <| 2> | 2 >>] fichier] [commande]

## Paramètres

| Paramètre | Détails |
|---|---|
| commander | Toute commande valide. |
| `>` | Ecrivez `STDOUT` dans un fichier. |
| `>>` | Ajouter `STDOUT` au fichier. |
| `<` | Lire le fichier sur `STDIN` . |
| `2>` | Ecrivez `STDERR` dans un fichier. |
| `2>>` | Ajouter `STDERR` au fichier. |
| fichier | Le chemin d'accès à un fichier. |

## Remarques

- Vous pouvez ajouter autant de redirections que vous le souhaitez, tant que le symbole et le fichier de redirection restent ensemble et dans le bon ordre.

## Examples

### Un exemple...

```
@echo off
setlocal
set /p "_myvar=what is your name?"
echo HELLO!>file.txt
echo %_myvar%!>>file.txt
echo done!
pause
type file.txt
endlocal
exit
```

Maintenant, file.txt ressemble à:

```
HELLO!
John Smith!
```

(en supposant que vous avez tapé `John Smith` comme votre nom.)

Maintenant, la console de votre fichier de commandes ressemble à:

```
what is your name?John Smith
done!
Press any key to continue...
HELLO!
John Smith!
```

(et il devrait sortir si rapidement que vous ne pourrez plus rien voir après l'invite `Press any key to coninue...` )

## Redirection du caractère spécial avec extension retardée activée

Cet exemple fait écho au caractère spécial `!` dans un fichier. Cela ne fonctionnerait que lorsque DelayedExpansion est désactivé. Lorsque l'extension différée est activée, vous devrez utiliser trois carets et un point d'exclamation comme ceci:

```
@echo off
setlocal enabledelayedexpansion


echo ^^^!>file
echo ^>>>file

goto :eof

    ^> is the text
    >> is the redirect operator

pause
endlocal
exit /b
```

Ce code fera écho au texte suivant dans le fichier

```
!
>
```

comme

```
^^^ escapes the ! and echos it into the file
^> escapes the > and echos it into the file
```

## Ecrire dans un fichier

```
@echo off
```

```
cls
echo Please input the file path, surrounded by "double quotation marks" if necessary.
REM If you don't want to redirect, escape the > by preceding it with ^
set /p filepath=^>

echo Writing a random number
echo %RANDOM% > %filepath%
echo Reading the random number
type %filepath%

REM Successive file writes will overwrite the previous file contents
echo Writing the current directory tree:
> %filepath% tree /A
echo Reading the file
type %filepath%

REM nul is a special file. It is always empty, no matter what you write to it.
echo Writing to nul
type %windir%\win.ini > nul
echo Reading from nul
type nul

echo Writing nul's contents to the file
type nul > %filepath%
echo Reading the file
type %filepath%
```

Lire Redirection d'entrée et de sortie en ligne: https://riptutorial.com/fr/batch-file/topic/7502/redirection-d-entree-et-de-sortie

# Chapitre 26: Si déclarations

## Syntaxe

- if [/ i] StringToCompare1 == StringToCompare2 (commandA) sinon (commandB)
- si errorlevel 1 (commandA) sinon (commandB)
- if% errorlevel% == 1 (commandA) sinon (commandB)
- si existe Nom de fichier (commandA) sinon (commandB)
- si défini NomVariable (commandA) sinon (commandB)

## Remarques

Il y a un peu de syntaxe à choisir dans une instruction `if` . Nous utiliserons `if string1==string2` comme exemple.

# Syntaxes 1 ligne

- `if string1==string2 commandA`
- `if string1==string2 (commandA)`

- `if string1==string2 (commandA) else (commandB)`
- `if string1==string2 (commandA) else commandB`

- `if string1==string2 (commandA)else (commandB)`
- `if string1==string2 (commandA)else commandB`

# Syntaxes multilignes

```
if string1==string2 (
    commandA
)
```

Ou

```
if string1==string2 (
    commandA
) else (
    commandB
)
```

Il y a encore des syntaxes supplémentaires disponibles.

# Examples

## Comparer des nombres avec un relevé IF

```
SET TEST=0

IF %TEST% == 0 (
    echo TEST FAILED
) ELSE IF %TEST% == 1 (
    echo TEST PASSED
) ELSE (
    echo TEST INVALID
)
```

## Comparaison de chaînes

```
IF "%~1" == "-help" (
    ECHO "Hello"
)
```

où `%1` fait référence au premier argument de ligne de commande et `~` supprime toutes les citations qui ont été inclus quand a été appelé le script.

## Comparer le niveau d'erreur

```
If Errorlevel 1 (
    Echo Errorlevel is 1 or higher

    REM The phrase "1 or higher" is used because If Errorlevel 1 statement means:
    REM                                          If %Errorlevel% GEQ 1
    REM                                      Not If %Errorlevel% EQU 1
)
```

ou

```
If "%Errorlevel%"=="1" (
    Echo Errorlevel is 1
)
```

Le script ci-dessus vérifie la variable Errorlevel (intégrée). L'opérateur `not` peut être utilisé.

```
Set "Test=%Errorlevel%"

If "%Test%" == "1" (
    Echo Errorlevel is 1
)
```

Celui-ci fonctionne également.

Veuillez noter que certaines commandes **n'affectent pas le niveau d'erreur** :

- Pause
- Écho
- Endlocal
- Pour
- Si
- Pause
- Rem
- Rd / Rmdir
- Ensemble
- Titre

Les commandes suivantes **définissent mais pas effacer errorlevel** :

- Cls
- Aller à
- Clés
- Popd
- Décalage

Les commandes suivantes **définissent les codes de sortie mais pas le niveau d'erreur** :

- Rd / Rmdir

Les commandes suivantes **définissent errorlevel mais pas les codes de sortie** :

- Md / Mkdir

### Vérifiez si le fichier existe

```
If exist "C:\Foo\Bar.baz" (
    Echo File exist
)
```

Cela vérifie si le fichier C: \ Foo \ Bar.baz existe. Si tel est le cas, echos File exist L'opérateur `Not` peut également être ajouté.

### Si la variable existe / set

```
If Defined Foo (
    Echo Foo is defined
)
```

Cela vérifierait si une variable est définie ou non. Encore une fois, l'opérateur `Not` peut être utilisé.

Lire Si déclarations en ligne: https://riptutorial.com/fr/batch-file/topic/5475/si-declarations

# Chapitre 27: Utiliser Goto

## Introduction

Aller est simple En utilisant des instructions simples, vous pouvez vous déplacer n'importe où dans votre code. Il peut également être utilisé pour créer des fonctions (montré dans la manière de créer des fonctions).

## Syntaxe

- goto: Label
- goto Label
- goto: EOF

## Paramètres

| Paramètre | Détails |
|---|---|
| `:Label` | Tout libellé valide (défini par `:<LabelName>` ) |
| `:EOF` | Une étiquette prédéfinie qui quitte le script actuel de la fonction (identique à `exit /b` ) |

## Remarques

En d'autres termes, si le numéro que le joueur a inséré est 1, il retournera à la partie: Nom du code.

donc si l'entrée est égale à 1, revenez à la ligne avec: Nom

Assurez-vous que si vous utilisez ceci, le mot commence par le Colen (:).

## Examples

### Exemples de programmes

Par exemple:

```
echo Hello!
pause >nul
:Name
echo What Is Your Name
set /p Input=Name:
echo so %Input% Is Your Name, right?
```

```
echo Rename?
echo 1 For Yes
echo 2 For No
set /p Input=Rename:
if %Input%=1 goto Name
```

Un autre exemple:

```
@echo off
echo 1 or 2?
set /p input=Choice:
if %input%=1 goto Skip
echo You Chose 1
pause >nul
echo So time for stuff
pause >nul
echo Random Stuf
pause >nul
:Skip
echo So that's it.
pause >nul
```

## Aller à variable

`Goto` accepte l'utilisation de la valeur de la variable pour agir en tant que label.

Exemple:

```
@echo off

echo a = 1
echo b = 2

set /p "foo=Enter option:"
goto %foo%
```

Cependant, vous devriez vérifier l'entrée afin qu'elle ne se déplace pas dans un endroit qui n'existe pas. Passer à une étiquette non définie mettra fin à votre script de lot instantanément.

Lire Utiliser Goto en ligne: https://riptutorial.com/fr/batch-file/topic/9164/utiliser-goto

# Chapitre 28: Variables dans des fichiers batch

## Examples

### Déclaration

Pour créer une variable simple et l'assigner à une valeur ou à une chaîne, utilisez la commande `SET` :

```
SET var=10
```

Ici, le code déclare une nouvelle variable `var` d'une valeur de `10` . Par défaut, toutes les variables sont stockées en interne sous forme de chaînes. cela signifie que la valeur `10` n'est pas différente de `foo1234` ou `Hello, World!`

# Notes sur les guillemets

Les guillemets utilisés seront inclus dans la valeur de la variable:

```
SET var="new value"             <-- %var% == '"new value"'
```

# Espaces dans les variables

Le langage par lots considère les espaces comme des parties acceptables des noms de variables. Par exemple, `set var = 10` donnera une variable appelée `var` contenant la valeur `10` (notez l'espace supplémentaire à droite de var et la gauche du 10).

# Utiliser des guillemets pour éliminer les espaces

Afin d'éviter les espaces, utilisez des guillemets autour de l'assignation entière; le nom et la valeur de la variable. Cela évite également les espaces de fin de ligne accidentels à la fin de la ligne (le caractère ␣ indique un espace):

```
SET␣var=my␣new␣value␣          <-- '%var%' == 'my new value '
SET␣"var=my␣new␣value"␣         <-- '%var%' == 'my new value'
```

Utilisez également des guillemets lorsque vous associez plusieurs instructions avec `&` ou `|` - sinon,

placez le symbole directement après la fin de la valeur de la variable:

```
SET var=val & goto :next        <-- '%var%' == 'val '
SET "var=val" & goto :next      <-- '%var%' == 'val'
SET var=val& goto :next         <-- '%var%' == 'val'
```

## Usage

```
echo %var%
```

Ce code fera écho à la valeur de `var`

Si `setLocal EnableDelayedExpansion` est utilisé, ce qui suit fera écho à la valeur de `var` (l'expression standard% var% ne fonctionnera pas dans ce contexte).

```
echo !var!
```

Dans les fichiers de commandes, les variables peuvent être utilisées dans n'importe quel contexte, y compris en tant que parties de commandes ou parties d'autres variables. Vous ne pouvez pas appeler une variable avant de la définir.

Utiliser des variables comme commandes:

```
set var=echo
%var% This will be echoed
```

Utilisation de variables dans d'autres variables:

```
set var=part1
set %var%part2=Hello
echo %part1part2%
```

## Substitution variable

Contrairement à d'autres langages de programmation, une variable est remplacée par sa valeur réelle dans un fichier de commandes **avant** l'exécution du script de traitement par lots. En d'autres termes, la substitution est effectuée lorsque le script est *lu* en mémoire par le processeur de commandes, et non lors de l' *exécution* ultérieure du script.

Cela permet d'utiliser des variables comme commandes dans le script, et dans le cadre d'autres noms de variables dans le script, etc. Le "script" dans ce contexte est une ligne ou un bloc de code, entouré de parenthèses: `()` .

Mais ce comportement signifie que vous ne pouvez pas modifier la valeur d'une variable dans un bloc!

```
SET VAR=Hello
FOR /L %%a in (1,1,2) do (
```

```
    ECHO %VAR%
    SET VAR=Goodbye
)
```

imprimera

```
Hello
Hello
```

car (comme vous le voyez, lorsque vous regardez le script exécuté dans la fenêtre de commande), il est évalué comme suit:

```
SET VAR=Hello
FOR /L %%a in (1,1,2) do (
    echo Hello
    SET VAR=Goodbye
)
```

Dans l'exemple ci-dessus, la commande `ECHO` est évaluée en tant que `Hello` lorsque le script est lu en mémoire, de sorte que le script fera écho à `Hello` pour toujours, même si de nombreux passages sont effectués via le script.

La manière d'obtenir le comportement de variable le plus "traditionnel" (de la variable en cours d'expansion pendant l'exécution du script) est de permettre une "extension retardée". Cela implique d'ajouter cette commande dans le script avant l'instruction de boucle (généralement une boucle FOR, dans un script de traitement par lots) et d'utiliser un point d'exclamation (!) Au lieu du signe de pourcentage (%) dans le nom de la variable:

```
setlocal enabledelayedexpansion
SET VAR=Hello
FOR /L %%a in (1,1,2) do (
    echo !VAR!
    SET VAR=Goodbye
)
endlocal
```

imprimera

```
Hello
Goodbye
```

La syntaxe `%%a in (1,1,2)` provoque l'exécution de la boucle 2 fois: à la première occasion, la variable porte sa valeur initiale «Hello», mais lors de la seconde, elle passe par la boucle - après avoir exécuté la seconde SET instruction comme la dernière action sur la 1ère passe - cela a changé à la valeur révisée «Au revoir».

**Substitution de variable avancée**

Maintenant, une technique avancée. L'utilisation de la commande `CALL` permet au processeur de commandes par lots d'étendre une variable située sur la même ligne du script. Cela peut fournir

une expansion à plusieurs niveaux, en utilisant CALL et l'utilisation de modificateurs.

Ceci est utile, par exemple, dans une boucle FOR. Comme dans l'exemple suivant, où nous avons une liste numérotée de variables:

```
"c:\MyFiles\test1.txt" "c:\MyFiles\test2.txt" "c:\MyFiles\test3.txt"
```

Nous pouvons y parvenir en utilisant la boucle FOR suivante:

```
setlocal enabledelayedexpansion
for %%x in (%*) do (
    set /a "i+=1"
    call set path!i!=%%~!i!
    call echo %%path!i!%%
)
endlocal
```

Sortie:

```
c:\MyFiles\test1.txt
c:\MyFiles\test2.txt
c:\MyFiles\test3.txt
```

Notez que la variable `!i!` est d'abord étendu à sa valeur initiale, 1, puis la variable résultante,% 1, est étendue à sa valeur réelle de `c:\MyFiles\test1.txt` . C'est la *double expansion* de la variable `i` . Sur la ligne suivante, `i` nouveau double, en utilisant la commande `CALL ECHO` avec le préfixe de la variable `%%` , puis imprimée à l'écran (c'est-à-dire affichée à l'écran).

A chaque passage successif de la boucle, le nombre initial est augmenté de 1 (à cause du code `i+=1` ). Ainsi, il augmente à `2` au 2ème passage dans la boucle et à `3` au 3ème passage. Ainsi, la chaîne qui résonne à l'écran se modifie à chaque passage.

## Déclarer plusieurs variables

Lorsque plusieurs variables sont définies au début du lot, une forme de définition courte peut être utilisée en utilisant une chaîne de remplacement .

```
@echo off
set "vars=_A=good,_B=,_E=bad,_F=,_G=ugly,_C=,_H=,_I=,_J=,_K=,_L=,_D=6
set "%vars:,=" & set "%"

for /f %%l in ('set _') do echo %%l
exit /b

_A=good
_D=6
_E=bad
_G=ugly
```

Notez que dans l'exemple ci-dessus, les variables sont triées en ordre alphabétique en mode natif, lorsqu'elles sont imprimées sur l'écran.

## Utiliser une variable comme tableau

Il est possible de créer un ensemble de variables pouvant agir de manière similaire à un tableau (bien qu'il ne s'agisse pas d'un objet de tableau réel) en utilisant des espaces dans l'instruction `SET` :

```
@echo off
SET var=A "foo bar" 123
for %%a in (%var%) do (
    echo %%a
)
echo Get the variable directly: %var%
```

Résultat:

```
A
"foo bar"
123
Get the variable directly: A "foo bar" 123
```

Il est également possible de déclarer votre variable à l'aide d'index afin de pouvoir récupérer des informations spécifiques. Cela créera plusieurs variables, avec l'illusion d'un tableau:

```
@echo off
setlocal enabledelayedexpansion
SET var[0]=A
SET var[1]=foo bar
SET var[2]=123
for %%a in (0,1,2) do (
    echo !var[%%a]!
)
echo Get one of the variables directly: %var[1]%
```

Résultat:

```
A
foo bar
123
Get one of the variables directly: foo bar
```

Notez que dans l'exemple ci-dessus, vous ne pouvez pas référencer `var` sans indiquer ce qu'est l'index souhaité, car `var` n'existe pas dans ses propres. Cet exemple utilise également `setlocal enabledelayedexpansion` conjointement avec les points d'exclamation à `!var[%%a]!` . Vous pouvez afficher plus d'informations à ce sujet dans la documentation de portée de substitution de variable .

## Opérations sur les variables

```
set var=10
set /a var=%var%+10
echo %var%
```

La valeur finale de `var` est 20.

La deuxième ligne ne fonctionne pas dans un bloc de commande utilisé, par exemple, sur une condition **IF** ou sur une boucle **FOR** , car une extension retardée serait nécessaire au lieu d'une extension de variable d'environnement standard.

Voici un autre moyen plus efficace de travailler dans un bloc de commandes:

```
set var=10
set /A var+=10
echo %var%
```

L'environnement d'invite de commandes prend en charge les valeurs entières signées sur 32 bits:

- addition `+` et `+=`
- soustraction `-` et `-=`
- multiplication `*` et `*=`
- division `/` et `/=`
- division de module `%` et `%=`
- bitwise ET `&`
- bit à bit OU `|`
- bitwise NON `~`
- binaire XOR `^`
- décalage binaire gauche `<<`
- décalage au niveau du bit `>>`
- logique non `!`
- unaire moins `-`
- grouper avec `(` et `)`

L'interpréteur de commandes Windows ne prend pas en charge les valeurs entières 64 bits ni les valeurs à virgule flottante dans les expressions arithmétiques.

**Remarque:** L'opérateur `%` doit être écrit dans un fichier de commandes sous la forme `%%` pour être interprété comme opérateur.

Dans une fenêtre d'invite de commandes exécutant le `set /A Value=8 % 3` lignes de commandes `set /A Value=8 % 3` la valeur `2` est `2` à la variable d'environnement `Value` et les sorties `2` .

Dans un fichier de commandes doit être écrit `set /A Value=8 %% 3` pour affecter la valeur `2` à la variable d'environnement `Value` et rien n'est sorti respectivement écrit pour gérer **STDOUT** (sortie standard). Un `set /A Value=8 % 3` lignes `set /A Value=8 % 3` dans un fichier de commandes entraînerait un message d'erreur *Opérateur manquant* lors de l'exécution du fichier de commandes.

L'environnement nécessite le commutateur `/A` pour les opérations arithmétiques uniquement, pas pour les variables de chaîne ordinaires.

Chaque chaîne de l'expression arithmétique après `set /A` si un numéro ou un opérateur est automatiquement interprété comme le nom d'une variable d'environnement.

Pour cette raison, en référençant la valeur d'une variable avec `%variable%` ou avec `!variable!` n'est pas nécessaire lorsque le nom de la variable ne comprend que des caractères verbaux (0-9A-Za-z_), le premier caractère n'étant pas un chiffre particulièrement utile dans un bloc de commandes commençant par `(` et se terminant par une correspondance `)` .

Les nombres sont convertis de chaîne en entier avec la fonction C / C ++ strtol, la `base` étant zéro, ce qui signifie que la détermination automatique de la base peut facilement donner des résultats inattendus.

Exemple:

```
set Divided=11
set Divisor=3

set /A Quotient=Divided / Divisor
set /A Remainder=Divided %% Divisor

echo %Divided% / %Divisor% = %Quotient%
echo %Divided% %% %Divisor% = %Remainder%

set HexValue1=0x14
set HexValue2=0x0A
set /A Result=(HexValue1 + HexValue2) * -3

echo (%HexValue1% + %HexValue2%) * -3 = (20 + 10) * -3 = %Result%

set /A Result%%=7
echo -90 %%= 7 = %Result%

set OctalValue=020
set DecimalValue=12
set /A Result=OctalValue - DecimalValue

echo %OctalValue% - %DecimalValue% = 16 - 12 = %Result%
```

La sortie de cet exemple est la suivante:

```
11 / 3 = 3
11 % 3 = 2
(0x14 + 0x0A) * -3 = (20 + 10) * -3 = -90
-90 %= 7 = -6
020 - 12 = 16 - 12 = 4
```

Les variables non définies lors de l'évaluation de l'expression arithmétique sont remplacées par la valeur 0.

## Définition de variables à partir d'une entrée

En utilisant le commutateur `/p` avec la commande `SET` vous pouvez définir des variables à partir d'une entrée.

Cette entrée peut être une entrée utilisateur (clavier):

```
echo Enter your name :
set /p name=
echo Your name is %name%
```

Ce qui peut être simplifié comme ceci:

```
set /p name=Enter your name :
echo Your name is %name%
```

Ou vous pouvez obtenir l'entrée d'un fichier:

```
set /p name=< file.txt
```

dans ce cas, vous obtenez la valeur de la première ligne à partir de `file.txt`

Obtenir la valeur de différentes lignes dans un fichier:

```
(
   set /p line1=
   set /p line2=
   set /p line3=

) < file.txt
```

Lire Variables dans des fichiers batch en ligne: https://riptutorial.com/fr/batch-file/topic/3528/variables-dans-des-fichiers-batch

# Crédits

| S. No | Chapitres | Contributeurs |
|---|---|---|
| 1 | Démarrer avec le fichier de commandes | BoeNoe, Clijsters, Community, DavidPostill, Derpcode, geisterfurz007, Jeffrey Lin, loadingnow, Mee, rudicangiotti, sambul35, Scott Beeson, Sourav Ghosh, stark, SteveFest, ths |
| 2 | Ajouter un délai au fichier batch | SteveFest, Tearzz, wizzwizz4 |
| 3 | Arguments de ligne de commande de fichier batch | DavidPostill, LotPings, npocmaka, sambul35 |
| 4 | Bogues dans le processeur cmd.exe | SteveFest, X. Liu |
| 5 | Changer de répertoire et lister son contenu | Aravind .KEN, SomethingDark, SteveFest, wizzwizz4 |
| 6 | Commandes batch obsolètes et leurs remplacements | npocmaka, SteveFest |
| 7 | Commentaires dans les fichiers batch | 0x90h, BoeNoe, DavidPostill, Gábor Bakos, JosefZ, LotPings, SachaDee, SomethingDark, Sourav Ghosh, Stephan, SteveFest , wasatchwizard |
| 8 | Contourner les limitations arithmétiques dans les fichiers de commandes | npocmaka |
| 9 | Création de fichiers à l'aide de lots | Aravind .KEN, David Starkey, fruitSalad266, J03L, LeoDog896, loadingnow, Tearzz |
| 10 | Différences entre lot (Windows) et terminal (Linux) | SomethingDark, SteveFest |
| 11 | Échapper des caractères spéciaux | npocmaka, SteveFest |

| | | |
|---|---|---|
| 12 | Écho | DavidPostill, fruitSalad266, Keith Hall, npocmaka, SomethingDark, Tearzz, wizzwizz4 |
| 13 | Fichiers aléatoires dans le lot | SteveFest |
| 14 | Fichiers batch et hybrides Powershell | npocmaka, SteveFest |
| 15 | Gestion des fichiers dans les fichiers de commandes | BoeNoe, LeoDog896, SteveFest |
| 16 | Hybrides Batch et JSCript | npocmaka, SteveFest |
| 17 | Hybrides batch et VBS | npocmaka, SteveFest |
| 18 | Les fonctions | ender_scythe, npocmaka, SteveFest |
| 19 | Les meilleures pratiques | SteveFest |
| 20 | Macros de fichiers par lots | SteveFest |
| 21 | Pile de répertoire | DavidPostill, wizzwizz4 |
| 22 | Pour les boucles dans les fichiers batch | David Starkey, DavidPostill, Ed999, Mee, sambul35, SomethingDark, SteveFest |
| 23 | Privilèges élevés dans les fichiers batch | DavidPostill, elzooilogico, sambul35 |
| 24 | Rechercher des chaînes dans un lot | sambul35 |
| 25 | Redirection d'entrée et de sortie | cascading-style, SomethingDark, SteveFest, wizzwizz4 |
| 26 | Si déclarations | npocmaka, Ozair Kafray, SomethingDark, SteveFest |
| 27 | Utiliser Goto | LeoDog896, SteveFest |
| 28 | Variables dans des fichiers batch | DavidPostill, Ed999, Jay, Jeffrey Lin, Mee, Mofi, SachaDee, sambul35, SomethingDark, SteveFest, Tearzz, ths, Toomaja, wasatchwizard, wizzwizz4 |