

# Bash : Les opérateurs de test sur fichiers

- Objet : Suite de la série de wiki visant à maîtriser bash via les caractères.
- Niveau requis : [débutant](#), [avisé](#)
- Commentaires : Bash, ligne de commande et scripts
- Débutant, à savoir : [Utiliser GNU/Linux en ligne de commande, tout commence là !](#) 😊
- Suivi :
  - [en-chantier](#), [à-tester](#), [à-placer](#)
    - Création par [agp91](#) 20/02/2023
    - Testé par <...> le <...> [Fix Me!](#)
- Commentaires sur le forum : [Lien vers le forum concernant ce tuto](#) <sup>1)</sup>
- [Vision d'ensemble](#)
- [Détail et caractères](#)
- [Les opérateurs de test sur paramètres](#)
- [Les opérateurs de test sur chaînes](#)
- **Les opérateur de test sur fichiers** 😊
- [Les opérateurs de comparaison numérique](#)
- [Les symboles dans les calculs](#)
- [Bash : les tableaux](#)
- [Les caractères de transformation de parametres](#)
- [Bash : Variables, globs étendus, ERb, ERe](#)

## Nota :

Contributeurs, les [Fix Me!](#) sont là pour vous aider, supprimez-les une fois le problème corrigé ou le champ rempli !

## Introduction



Dans la page du manuel de bash, **les opérateurs des commandes de test** sont nommées **primitives**.

Bash dispose de plusieurs commandes pour **réaliser des tests sur des fichiers**.

- Les commandes internes **[** et **test**.
- Et la commande composée **[[**.



- Les commandes **[** et **test** sont équivalentes.
- Les commandes **[** et **test** sont disponibles dans leurs versions externe : **/usr/bin/[** et **/usr/bin/test**.
  - Elles ont toutes les deux la même page de manuel (**man [** ou **man test**).
- Les commandes internes disposent de primitive que n'ont pas les commandes externes.

### Rappels :



- Une commande de test renvoie le code de retour 0 (considérer comme vrai) lorsque le test réussi et 1 (considérer comme faux) lorsqu'il échoue.
- Le code retour d'une commande est mémorisé dans le paramètre spécial \$?.
- L'opérateur de contrôle **&&** exécute la commande suivante, si la commande précédente à renvoyée un code de retour égale à 0.
- L'opérateur de contrôle **||** exécute la commande suivante, si la commande précédente à renvoyée un code de retour supérieur à 0.

Les commandes de test disposent de 24 primitives ( 21 unaires et 3 binaires) pour tester les fichiers.

## Synopsis

- **test OP fichier**
- **[ OP fichier ]**
- **[[ OP fichier ]]**
  
- **test fichier1 OP fichier2**
- **[ fichier1 OP fichier2 ]**
- **[[ fichier1 OP fichier ]]**
  
- Avec :
  - **Fichier, fichier1** et **fichier2** sont des chemins de fichier. Ils sont sujet aux développements.
  - **OP**, l'une des primitives suivante :

Liste des primitives de test sur les fichiers	
Primitives	Retours
<b>-a fichier</b>	Vrai si <b>fichier</b> existe.
<b>-b fichier</b>	Vrai si <b>fichier</b> est un fichier spécial bloc.
<b>-c fichier</b>	Vrai si <b>fichier</b> est un fichier spécial caractère.
<b>-d fichier</b>	Vrai si <b>fichier</b> existe et est un répertoire.
<b>-e fichier</b>	Vrai si <b>fichier</b> existe.
<b>-f fichier</b>	Vrai si <b>fichier</b> est un fichier normal.
<b>-g fichier</b>	Vrai si <b>fichier</b> a son bit Set-GID positionné.
<b>-h fichier</b>	Vrai si <b>fichier</b> est un lien symbolique.
<b>-k fichier</b>	Vrai si <b>fichier</b> a son bit « sticky » positionné.
<b>-p fichier</b>	Vrai si <b>fichier</b> est un tube nommé (FIFO).
<b>-r fichier</b>	Vrai si <b>fichier</b> est accessible en lecture.
<b>-s fichier</b>	Vrai si <b>fichier</b> a une taille strictement positive.
<b>-t df</b>	Vrai si le descripteur de fichier <b>df</b> est ouvert et se rapporte à un terminal.
<b>-u fichier</b>	Vrai si <b>fichier</b> a son bit Set-UID positionné.
<b>-w fichier</b>	Vrai si <b>fichier</b> est accessible en écriture.
<b>-x fichier</b>	Vrai si <b>fichier</b> est exécutable.

<b>-G fichier</b>	Vrai si <b>fichier</b> appartient au GID effectif du groupe.
<b>-L fichier</b>	Vrai si <b>fichier</b> est un lien symbolique.
<b>-N fichier</b>	Vrai si <b>fichier</b> a été modifié depuis sa dernière lecture.
<b>-O fichier</b>	Vrai si <b>fichier</b> appartient à l'UID effectif de l'utilisateur.
<b>-S fichier</b>	Vrai si <b>fichier</b> est une socket.
<b>fichier1 -ef fichier2</b>	Vrai si <b>fichier1</b> et <b>fichier2</b> se rapportent au même périphérique et ont les mêmes numéros d'inœuds.
<b>fichier1 -nt fichier2</b>	Vrai si <b>fichier1</b> est plus récent que <b>fichier2</b> ou si <b>fichier1</b> existe et non <b>fichier2</b> .
<b>fichier1 -ot fichier2</b>	Vrai si <b>fichier1</b> est plus ancien que <b>fichier2</b> ou si <b>fichier2</b> existe et non <b>fichier1</b> .



La primitive **-e** teste si le fichier existe qu'il soit un fichier normal, spécial caractère, spécial bloc, un répertoire, un lien symbolique ou une socket.

## Exemples

Testons si le fichier **/tmp/toto** existe.

```
touch /tmp/toto          # Crée le fichier /tmp/toto.

test -f /tmp/toto      ; echo $?    # Test si le fichier /tmp/toto est un
fichier.
[ -d /tmp/toto ]      ; echo $?    # Test si le fichier /tmp/toto est un
répertoire.

rm -v /tmp/toto        # Supprime le fichier /tmp/toto.
test -f /tmp/toto      ; echo $?    # Test si le fichier existe.
```

```
0
1
0
'/tmp/toto' supprimé
```

Testons le développement du **~** (tilde) avec le fichier **~/toto**.

En copiant le code ci-dessous, dans le fichier **mon\_script**.

Puis exécutons le, avant de le supprimer.

script

```
#!/bin/bash
var1="~/toto"
touch $var1

if [ -f "$var1" ]
then
```

```
echo "Le fichier existe !"
else
  echo "Fichier non trouvé ! Car il n'y a pas de développement du ~
entre guillemets (simples ou doubles)."
fi

var1=~ /toto # Sans guillemets, le ~ est développé.
if [ -f "$var1" ]
then
  echo "Le fichier existe ! (Sans guillemets, le développement du ~
c'est réalisé)."
else
  echo "Fichier inexistant !"
fi

rm ~/toto
```

```
bash mon_script
```

```
rm -v mon_script
```

```
Fichier non trouvé ! Car il n'y a pas de développement du ~ entre guillemets
(simples ou doubles).
Le fichier existe ! (Sans guillemets, le développement du ~ c'est réalisé).
'mon_script' supprimé
```

Testons ce qu'est **/dev/null**.

```
f=/dev/null
ok="$f est"
ko="$f n'est pas"

test -e $f && echo "$f existe." || echo "$f n'existe pas."
[ -r $f ] && echo "$ok en lecture." || echo "$ko en lecture"
[[ -w $f ]] && echo "$ok en écriture." || echo "$ko en ecriture."
[ -x $f ] && echo "$ok exécutable." || echo "$ko exécutable."
ok="$ok un"
ko="$ko un"
[ -d $f ] && echo "$ok répertoire." || echo "$ko répertoire."
[ -h $f ] && echo "$ok lien." || echo "$ko lien symbolique."
[ -S $f ] && echo "$ok socket." || echo "$ko socket."
ok="$ok fichier"
ko="$ko fichier"
[ -f $f ] && echo "$ok normal." || echo "$ko normal."
[ -b $f ] && echo "$ok spécial bloc." || echo "$ko spécial bloc."
[ -c $f ] && echo "$ok spécial caractère." || echo "$ko spécial caractère."
```

```
unset f ok ko
```

```
/dev/null existe.  
/dev/null est en lecture.  
/dev/null est en écriture.  
/dev/null n'est pas exécutable.  
/dev/null n'est pas un répertoire.  
/dev/null n'est pas un lien symbolique.  
/dev/null n'est pas un socket.  
/dev/null n'est pas un fichier normal.  
/dev/null n'est pas un fichier spécial bloc.  
/dev/null est un fichier spécial caractère.
```

Et oui **/dev/null** est un fichier spécial caractère, accessible en lecture et en écriture. Certains le nomme le puits ou encore le trou (noir)... Mais “devnull” lui va si bien ;) En tout cas, tous ce qui y va, n'en ressort plus et est perdu à jamais.

Et voilà ! 😊

## Tuto précédent

[Les opérateurs de test sur chaînes](#)

## La suite c'est ici

[Les opérateurs de comparaison numérique](#)

1)

N'hésitez pas à y faire part de vos remarques, succès, améliorations ou échecs !

From:

<http://debian-facile.org/> - **Documentation - Wiki**

Permanent link:

<http://debian-facile.org/atelier:chantier:bash:les-operateurs-de-test-sur-fichiers>



Last update: **21/02/2023 11:45**