Tabel II: Bestand testen

Expressie	Is waar als bestand
-f bestand	een gewone file is (dus geen directory of block device bijvoorbeeld)
-d bestand	een directory is
-L bestand	een symlink is
-x bestand	uitvoerbaar is door de gebruiker van het script
-s bestand	niet leeg is

bedoeld. De commando's **test** en **[** evalueren de vergelijking. Is die waar, dan is de returncode 0 en gaat het script verder met de then-tak. In overige gevallen is de returncode niet gelijk aan 0 en gaat de if-constructie de else-tak in.

Daarom kun je het script bondig schrijven als:

```
#!/bin/bash

if ls /root ; then
        echo "Je hebt wel root-rechten"
else
        echo "Je hebt geen root-rechten"
fi
```

Dit passen we nu toe in ons script, maar dan voor de while-loop.

BESTAND INLEZEN

Zet de lijst van woorden in een bestand lijst.txt:

```
cat lijst.txt
linux
maGAzine
```

Vervolgens maken we een script dat als invoerargument de naam van het bestand meekrijgt:

```
#!/bin/bash

lijst=$1
```

Om de inhoud regel voor regel in te lezen, voeg je een while-loop toe:

```
while read woord ; do
        woord=${woord,,}
        echo ${woord^}
done < "$lijst"
```

Het commando **read** leest één regel van standaard invoer in en geeft dat als waarde aan de variabele woord. De while-loop bepaalt vervolgens aan de hand van de returncode van **read** of hij de commando's moet uitvoeren. **Read** geeft altijd returncode 0, tenzij hij end-of-file tegenkomt. Op de commandoregel doe je dat door de toetsencombinatie Ctrl+D. In dit voorbeeld bereik je dat door het inlezen van een bestand tot het einde.

Het inlezen gebeurt aan het eind van de while-loop. Daar zie je door het kleiner-dan-teken dat het bestand als invoer dient. Na iedere read schuift de while-loop een regel verder op in het bestand. De dubbele aanhalingstekens zijn toegevoegd in het geval je een bestandsnaam met spaties gebruikt.

Dit werkt overigens niet:

```
while read woord  < "$lijst" ; do
        woord=${woord,,}
        echo ${woord^}
done
```

Read leest de eerste regel van het bestand en stopt. Dat wil zeggen: hij sluit het bestand. Vervolgens zie je de eerste regel op het scherm en voert de while-loop het **read** commando opnieuw uit. Die leest diezelfde eerste regel in en stopt weer. Je ziet de eerste regel voor de tweede keer en de cyclus begint opnieuw. Kortom, je hebt een eeuwige loop.

Je gebruikt dit script als volgt:

```
./mijn_script lijst.txt
Linux
Magazine
```

Ook bij je eigen script is redirectie gewoon te gebruiken:

```
./mijn_script lijst.txt >mijn_uitvoer.txt
```

Bedenk hoe je dit in het script regelt. Tip: gebruik redirectie van het echo commando.

BESTAND TESTEN

Bij een verkeerde bestandsnaam mislukt het script domweg:

```
./mijn_script niet_bestaand.txt
./mijn_script: line 8: niet_bestaand.txt: No such file or directory
```

Voor het netjes afhandelen van deze foutsituatie moeten we controleren of het bestand bestaat. In tabel II zie je veelgebruikte testmogelijkheden op bestanden. Voeg de volgende test vóór de while-loop toe:

```
if [ ! -f "$lijst" ] ; then
        echo Bestand $lijst bestaat niet
        exit 1
fi
```

Eigenlijk is **-f** een optie bij het commando testen. Dat geldt ook voor de andere testmogelijkheden in tabel II. Door het uitroepteken draaien we het resultaat van de evaluatie om. Daardoor gaan we de then-tak in, indien het bestand juist niet bestaat.

Toch is het script niet helemaal correct. Voer het als volgt uit:

```
./mijn_script niet_bestaand.txt >mijn_uitvoer.txt
```

Nu zie je geen foutmelding, want die staat door de redirectie in mijn_uitvoer.txt. Daarom is er behalve standaarduitvoer ook uitvoer specifiek voor foutmeldingen.

STANDAARD ERROR

Om gewone uitvoer te onderscheiden van foutmeldingen heb je het standaard error kanaal (zie afbeelding 3). Ook dat doe je met redirectie. De syntax hiervan oogt behoorlijk cryptisch:

```
echo Foute boel >&2
```

Het echo commando stuurt de melding naar standaard uitvoer. Die leiden we vervolgens om door het groter-dan-teken naar het standaard error **kanaal &2**. Voluit is het commando eigenlijk:

```
echo Foute boel 1>&2
```

Hierbij staat **1** voor standaard uitvoer. Zo zie je expliciet dat je gewone uitvoer omleidt. De ampersand ( ) geeft aan dat het niet om redirectie naar een gewoon bestand gaat. Dit:

```
echo Foute boel 1>2
```

Levert het bestand 2 op:

```
cat 2
Foute boel
```

Pas in het script het echo commando aan:

```
echo Bestand $lijst bestaat niet >&2
```

Voer het script opnieuw uit:

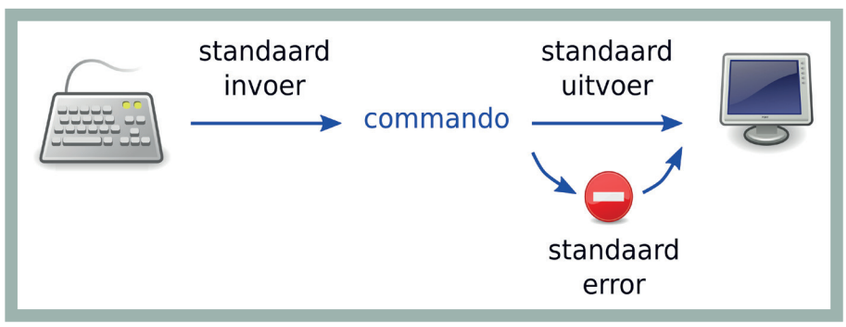
```
./mijn_script niet_bestaand.txt >mijn_uitvoer.txt
```

Nu zie je de foutmelding wel keurig op je scherm. Tegelijkertijd blijft je uitvoerbestand ervan verschoond, omdat die alleen de gewone uitvoer ontvangt.

Voor de volledigheid melden we nog dat standaard invoer het getal 0 heeft. Die zie je maar zelden in scripts.

STRINGS VERGELIJKEN

In de while-loop van hierboven manipuleer je elk woord op dezelfde manier. Om onderscheid te maken moet je strings vergelijken. In deel 1 zag je hoe je dat deed voor getallen.



Afbeelding 3: Foutmeldingen naar error kanaal