Tabel I: Strings bewerken

Bewerking	Omschrijving	Voorbeeld	
		woord=MagaZine	resultaat
${woord:positie}	Toon waarde vanaf positie (begint bij 0)	echo ${woord:1}	agaZine
${woord:positie:lengte}	Toon waarde vanaf positie met aantal karakters gelijk aan lengte	echo ${woord:1:4}	agaZ
${woord#match}	Verwijder kortste match aan begin	echo ${woord#*a}	gaZine
${woord##match}	Verwijder langste match aan begin	echo ${woord##*a}	Zine
${woord%match}	Verwijder kortste match aan eind	echo ${woord%a*}	Mag
${woord%%match}	Verwijder langste match aan eind	echo ${woord%%a*}	M
${woord^}	Eerste letter wordt hoofdletter	echo ${woord^}	MagaZine
${woord^^}	Alle letters worden hoofdletter	echo ${woord^^}	MAGAZINE
${woord,}	Eerste letter wordt kleine letter	echo ${woord,}	magaZine
${woord,,}	Alle letters worden kleine letter	echo ${woord,,}	magazine

```
i=0
while [ $# -ne 0 ] ; do
        woorden[$i]=$1
        shift
        i=$(($i + 1))
done
```

Het is niet nodig om het array niet te initialiseren door **woorden=()**. Bij het toevoegen van het eerste element definieert Bash automatisch een array.

Het aantal elementen in de array vind je door:

```
${#woorden[@]}
```

Het apenstaartje haalt alle elementen op en de hash (**#**) telt vervolgens het aantal. Voeg nu dit toe aan het script om alle woorden te tonen:

```
i=0
while [ $i -lt ${#woorden[@]} ] ; do
        echo ${woorden[$i]}
        i=$(($i + 1))
done
```

De teller i begint bij 0, dat wil zeggen: het eerste woord. Die toon je en vervolgens hoog je de teller op. Dit gaat door, totdat de teller gelijk wordt aan het aantal elementen in de array. Daardoor faalt de test en stopt de loop.

Beredeneer waarom de test niet pas mag falen als de teller groter is dan het aantal elementen:

```
while [ $i -le ${#woorden[@]} ]
```

Tip: bepaal hoe vaak de while-loop al gelopen heeft, als in de test de teller en het aantal woorden beide gelijk zijn aan 2.

STRINGS BEWERKEN

In tabel I zie je een selectie van de vele mogelijkheden om strings te bewerken. Als voorbeeld passen we de while-loop aan, zodat elk woord met een hoofdletter begint, gevolgd door kleine letters. Eerst zet je het volledige woord om naar kleine letters. Daarna maak je van de eerste letter een hoofdletter:

```
i=0
while [ $i -lt ${#woorden[@]} ] ; do
        woord=${woorden[$i],,}
        echo ${woord^}
        i=$(($i + 1))
done
```

De uitvoer hiervan:

```
./mijn_script linux maGAzine
Linux
Magazine
```

Stringmanipulatie werkt ook met arrays. Bijvoorbeeld:

```
mijn_array=(a b c)
```

Maak van het tweede element een hoofdletter:

```
mijn_array[1]=${mijn_array[1]^}
echo ${mijn_array[@]}
a B c
```

Ook nu manipuleer je met het apenstaartje alle elementen in één keer:

```
hoofdletters=${mijn_array[@]^}
echo ${hoofdletters[@]}
A B C
```

REDIRECTIE

Redirectie betekent dat je in- of uitvoer omleidt (zie afbeelding 2). Om de uitvoer van een echo commando weg te schrijven naar een bestand gebruik je > (het groter-dan-teken):

```
echo Linux > mijn_bestand
```

De spatie na het groter-dan-teken mag je weglaten. De inhoud van **mijn_bestand** is nu:

```
cat mijn_bestand
Linux
```

Bij een volgend woord overschrijf je de inhoud:

```
echo Magazine > mijn_bestand
cat mijn_bestand
Magazine
```

Om het aan bestaande inhoud toe te voegen, gebruik je het teken dubbel:

```
echo Linux > mijn_bestand
echo Magazine >> mijn_bestand
cat mijn_bestand
Linux
Magazine
```

Met < (het kleiner-dan-teken) zorg je dat een bestand als invoer dient voor een commando:

```
sort < mijn_bestand
```

Beide vormen van redirectie kun je gewoon combineren:

```
sort < mijn_bestand > mijn_gesorteerd_bestand
```

Afbeelding 2: In- en uitvoer omleiden

Het weglaten van de spaties is hier wel duidelijker:

```
sort <mijn_bestand >mijn_gesorteerd_bestand
```

Dadelijk gebruiken we dit om de woordenlijst uit een bestand te lezen, maar eerst nog even iets over testen.

NOGMAALS DE RETURNCODE

In deel 1 zag je hoe je met de returncode het al dan niet succesvol verlopen van een commando test. Daartoe vergelijk je de returncode met het cijfer **0**:

```
#!/bin/bash

ls /root

if [ $? -eq 0 ] ; then
        echo “Je hebt wel root-rechten”
else
        echo “Je hebt geen root-rechten”
fi
```

Eigenlijk test de if-constructie de returncode van een commando. Dat commando is de eerste vierkante haak en die vind je zelfs op je systeem:

```
which [
/usr/bin/[
```

Die haak is een synoniem voor het commando test. Dit doet dan ook hetzelfde:

```
if test $? -eq 0 ; then
```

De sluithaak bij **[** is een vereist argument, maar puur syntactisch