Universiteit van Amsterdam

Opgave 2

Jouke Witteveen, Bas Terwijn

Back

Deadline: Individueel: Woensdag 6 februari 2019 23:59
Team: zondag 10 Februari 2019, 23:59

1 Introductie en puzzeltjes (ontwikkelend)

We beginnen met enkele kleine opgaven om beter bekend te raken met bash. Daarna volgt de individuele opgave, welke individueel gemaakt moet worden, en vervolgens de opgave 'Back'.

Opgave 1

Implementeer een begroetingsprogramma dat de gebruiker vraagt naar zijn naam en de gebruiker begroet.

```
$ ./greeter1.sh
What is your name?
Jouke  <-- deze regel wordt ingevoerd in het programma en maakt geen deel uit van de uitvoer
Hello Jouke!
```

Opgave 2a

Verbeter uw begroeting zodat lege invoer genegeerd wordt.

```
$ ./greeter2a.sh
What is your name?
       <-- lege invoer
       <-- meer lege invoer
Jouke  <-- niet-lege invoer
Hello Jouke!
```

Opgave 2b

Zorg dat de begroeting blijft herhalen totdat de gebruiker `quit` invoert.

```
$ ./greeter2b.sh
What is your name?
       <-- lege invoer
Jouke  <-- niet-lege invoer
Hello Jouke. Next!
What is your name?
quit   <-- quit invoer
Ok, bye.
```

Opgave 3a

Pas de begroeting aan zodat alle klinkers worden vervangen door underscores.

```
$ ./greeter3a.sh
What is your name?
Jouke
Hello J__k_!
```

Opgave 3b

Vervang in plaats van alle klinkers nu alleen alle exemplaren van het laatste karakter door underscores.

```
$ ./greeter3b.sh
What is your name?
Witteveen, Jouke
Hello Witt_v__n, Jouk_!
```

Opgave 4a (competent)

Pas de begroeting aan en geef alle regels in bestand '/etc/passwd' waar *niet* de naam van de gebruiker in voorkomt, ongeacht klein- of hoofdlettergebruik. Groet de gebruiker vervolgens alleen als de naam is voorgekomen.

Lees daarbij elke regel in '/etc/passwd' slechts 1 keer en gebruik niet het shell commando 'grep'.

```
$ ./greeter4a.sh
What is your name?
Jouke         <-- dit is invoer, het onderstaande is uitvoer afkomstig van '/etc/passwd'
root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/usr/bin/nologin
daemon:x:2:2:daemon:/:/usr/bin/nologin
[...]         <-- meer regels, maar dus zonder "jouke" of "JOUke" of ...
Hello Jouke!  <-- alleen als Jouke voorkwam
```

Opgave 4b (competent)

Implementeer hetzelfde als bij opgave 4a maar nu met gebruik van `grep`. Meerdere keren '/etc/passwd' lezen is toegestaan. Noem deze file `greeter4b.sh`

Opgave Individueel

Schrijf een `Bash` script dat gegevens over personen in bestand *persons_small.txt* met dit formaat:

```
  Rebekah Kirsche / personality : Hardworking, Careless /
employment:Cartographer /birth day : 1983-12-7 |Joycelyn Loetstein
 /mothers name : Tscharke / personality : Perceptive, Patronizing,
Trustworthy / employment : Oceanographer |Geno Palomares Morstedt /
employment : Stockbroker / ID number : 733265098820939 / mothers
name: Escarate | Shaida Boms /birth day : 1985-1-8 / ID
number : 43298815757784/ mothers name : Kruch /relationship status :
domestic partnership
```

om kan zetten in bestand *persons_small.xml* met xml formaat:

```
<?xml version="1.0" encoding="UTF-8"?>
<persons>
  <person name="Rebekah Kirsche">
    <personality>Hardworking, Careless</personality>
    <employment>Cartographer</employment>
    <birth_day>1983-12-7</birth_day>
  </person>
  <person name="Joycelyn Loetstein">
    <mothers_name>Tscharke</mothers_name>
    <personality>Perceptive, Patronizing, Trustworthy</personality>
```

```
    <employment>Oceanographer</employment>
  </person>
  <person name="Geno Palomares Morstedt">
    <employment>Stockbroker</employment>
    <ID_number>733265098820939</ID_number>
    <mothers_name>Escarate</mothers_name>
  </person>
  <person name="Shaida Boms">
    <birth_day>1985-1-8</birth_day>
    <ID_number>43298815757784</ID_number>
    <mothers_name>Kruch</mothers_name>
    <relationship_status>domestic partnership</relationship_status>
  </person>
</persons>
```

Het invoer-formaat begint met een 'naam' van een persoon dan een `'/'` scheidingsteken gevolgd door een lijst met attributen gescheiden door het `'/'` scheidingsteken. Elk attribuut bestaat uit een 'key' dan een `':'` en dan de 'value' die bij de 'key' hoort. Dit patroon herhaalt zich voor alle personen in dit bestand waarbij personen worden gescheiden door het `'|'` scheidingsteken. In het xml bestand zijn alle overbodige whitespaces verwijderd en omdat xml tags geen whitespaces mogen hebben is daar iedere resterende whitespace vervangen door een underscore.

Gebruik alleen `Bash`-eigen commando's, dus geen `awk`, `grep`, `sed`, of andere shell commando's. Test je `Bash` script door het grote bestand *persons_large.txt* om te zetten naar xml en deze met de `diff` tool te vergelijken met *persons_large.xml*.

Hints and tips

Bij word splitting gebruikt `Bash` de characters in de 'IFS' variabele als seperators. De default waarde van IFS is whitespace, tab en newline, maar we kunnen zelf onze eigen seperator characters kiezen, een voorbeeld:

```
test_string="aa bb , cc dd ; ee ff , gg"
IFS=',;'
for s in $test_string; do
    echo "'$s'"
done

read -r first second tail <<< $test_string
echo "first: '$first'"
echo "second: '$second'"
echo "tail: '$tail'"
```

```
'aa bb '
' cc dd '
' ee ff '
' gg'
first: 'aa bb '
second: ' cc dd '
tail: ' ee ff , gg'
```

Command 'read' leest bij elke aanroep data tot de delimiter welke bij default het newline character is. Met '-d' kunnen we zelf de delimiter kiezen, bijvoorbeeld zodat we meer dan 1 regel tegelijk kunnen inlezen. Wanneer in de gelezen data geen delimiter is gevonden (omdat de data op is) geeft het een 'failure' (non-zero) status terug. Dit kan zorgen dat een loop eindigt zonder dat we de laatste data hebben verwerkt. Maar, als we in de loop-conditie ook testen of de gelezen data niet een lege string is, hebben we daar geen last van, een voorbeeld:

```
test_string="aa bb , cc dd , ee ff , gg"
while read -r -d ',' block || [[ -n "$block" ]]; do
    echo "block: '$block'"
done <<< $test_string
```

```
block: 'aa bb'
block: 'cc dd'
block: 'ee ff'
block: 'gg'
```

Opgave Back

Het doel van het spel is om een string te transformeren die bestaat uit `a` s, `b` s en `c` s in een andere string door een reeks van zetten. Elke zet heeft bijbehorende kosten en de totale kosten van de transformatie moeten worden geminimaliseerd.

Zet (competent)

Het spel heeft slechts één type zet. In een zet wordt een blok van opeenvolgende en gelijke tekens geselecteerd in de string. Dit blok wordt uit de string geknipt, de tekens van het blok worden een plaats geroteerd in het alfabet (`a` → `b` → `c` → `a`), en daarna wordt het blok achteraan aan de string geplakt. Dus, het selecteren van het middenblok van `b` s in `aabbaa` transformeert de string in `aaaacc`:

aa<u>bb</u>aa		(origineel)
aaaa	bb	(knip)
aaaa	cc	(roteer)
aaaa<u>cc</u>		(plak)

Terwijl het selecteren van het blok `a` in `abc` de string in `bcb` transformeert:

<u>a</u>bc		(origineel)
bc	a	(knip)
bc	b	(roteer)
bc<u>b</u>		(plak)

Addressering (gevorderde)

Het selecteren van een blok gebeurt met behulp van een adres-string. Aanvankelijk is de zoekopdracht verankerd aan de uiterste linkerkant van de string. Het eerste karakter van de adres-string verplaatst het anker naar rechts, naar de positie waar dit karakter als eerste voorkomt. Dat blok van opeenvolgende en gelijke karakters is nu geselecteerd. Dus de adresstring `a` kan worden gebruikt om het eerste blok `aa` in `aabbaa` te selecteren:

<u>aa</u>bbaa

En de adres-string `b` kan worden gebruikt om het eerste blok `bb` te selecteren:

aa<u>bb</u>aa

Het volgende karakter in de adres-string verplaatst het anker verder naar rechts, eerst naar het einde van het blok en dan naar de positie waar dat karakter als eerste voorkomt. Nadat elk teken van de adres-string is verwerkt, bepaalt de uiteindelijke positie van het zoekanker welk blok van opeenvolgende en gelijke karakters is geselecteerd. Dus in string `aabbccaabbcc` selecteert adres-string `b`:

aa<u>bb</u>ccaabbcc

adres-string `ba` en `aa` selecteren beide:

aabbcc<u>aa</u>bbcc

en adres-string `cc` en `bac` selecteren beide:

aabbccaabb<u>cc</u>

In string `aabbaa` selecteert adres-string `c` en `bb` een leeg blok.

Kosten (vaardig)

De kosten van een zet zijn exponentieel in de lengte van adres-string, waar de exponentiele wordt genomen met basis 2. Dus, de kosten van een zet gespecificeerd door `a` (lengte:1) is 2, terwijl de kosten van een zet gespecificeerd door `aa` (lengte:2) 4 is, en die van `aba` (lengte:3) is 8.

Missie

Een missie bestaat uit een lijst met spelstrings, waarbij elke string moet worden getransformeerd in de volgende. Elke succesvolle transformatie verdient de speler een budget dat moet worden gebruikt voor de volgende transformaties. Wanneer de laatste transformatie is gemaakt, is de missie voltooid en het resterende budget zal de score van de speler zijn. Wanneer de speler tijdens het spelen onvoldoende budget heeft om nog een zet te doen, is het spel voorbij en is de missie mislukt.

Missies worden gespecificeerd door een op tekst gebaseerd missiebestand, waarbij elke regel van het bestand bestaat uit een door spaties gescheiden string en een nummer. De eerste regel geeft de beginstring en het begin-budget. Elke volgende regel geeft de volgende doelstring om naar toe te transformeren en het budget dat wordt verdiend bij een succesvolle transformatie. De doel-regels van de missie moeten worden bereikt in de volgorde zoals gegeven in het missiebestand. Zie bestand *mission*.

Het pad naar het missiebestand wordt gegeven als eerste argument aan het **back**-programma.

Save game (gevorderde)

Wanneer een mission file erg lang is, kan het zijn dat je het spel wilt afbreken en het later verder wilt spelen. De taak is om een commando `save` te implementeren, dat de huidige spel state wegschrijft naar een file genaamd *.save*. Voor een missiebestand dat is opgeslagen in */path/mission*, moet het save bestand worden opgeslagen in */path/.mission.save*. Als de mission file opnieuw geopend word en er is een save file aanwezig, moet de speler verder kunnen gaan waar hij was.

Voor het expert niveau is het ook de bedoeling dat je het spel kan afsluiten doormiddel van `ctrl-c` en dan de huidige spel state wegschrijft doormiddel van het hierboven beschreven commando.

Voorbeeld

Stel we beginnen met beginstring `abcabc`, een begin-budget van 10 en we doen onderstaande zetten:

`abcabc`	(budget:10)
zet: `b`	(cost:2)
`acabcc`	(budget:8)
zet: `cc`	(cost:4)
`acabaa`	(budget:4)

De totalen kosten van de zetten zijn 6. Als het begin-budget minder dan 6 was geweest, dan was deze transformatie niet mogelijk geweest.

Opgave

Implementeer het spel van **back** in `Bash`, zodat het door een mens kan worden gespeeld. Dit betekent dat je tijdens een spel bij elke zet een duidelijke weergave met alle relevante informatie geeft. Niettemin, de leesbaarheid van uw broncode is belangrijker dan een fancy weergave. Gebruik alleen `Bash`-eigen commando's, dus geen `awk`, `grep`, `sed`, of andere shell commando's.

Het bestand *skel.sh* bevat broncode om missie bestanden in te lezen.

Implementeer voor het expert niveau voor deze opgave ook een scorebord dat de vier hoogste scores bijhoudt van een missie. Voor een missiebestand dat is opgeslagen in */path/mission*, moet het scorebord worden opgeslagen in */path/.mission.leaderboard*. De regels in het leaderboard-bestand moeten bestaan uit een spatie-gescheiden getal en string, die respectievelijk een score en een naam vertegenwoordigen. Een naam kan op zijn beurt ook spaties bevatten. Het leaderboard-bestand moet worden bijgehouden in volgorde van afnemende scores.

2 Inleveren

Voor de individuele deadline lever alleen `individual.sh` in, voor de team deadline lever alle `greeter*.sh` files in en de `back.sh`