COMPUTERGEBRUIK: **oefeningenreeks 2** *2014-2015*

Pictogrammen: Bepaalde oefeningen maken deel uit van de opgelegde oefeningen die tijdens de evaluaties bevraagd kunnen worden. De volgende tabel geeft een overzicht van de betekenis van de gebruikte pictogrammen:

- **B** Basisvraag. Introduceert nieuwe technieken
- ★ Verplichte oefening die tijdens de evaluaties aan bod kan komen.
- **E** Vroegere examenvraag.

Bijkomende examenvragen van vorige jaren zijn beschikbaar op Minerva.

Evaluatie: Tijdens de werkcolleges mogen de studenten onderling van gedachten wisselen wanneer ze hun oefeningen aan het afwerken zijn. Onthoud echter dat wat je zelf vindt, veel langer zal blijven hangen, en wellicht tijd zal besparen tijdens het examen. De volgende planning wordt vooropgesteld:

06/11	werkcollege 6	*27/11*	werkcollege 9
13/11	werkcollege 7	*04/12*	werkcollege 10
20/11	werkcollege 8	*11/12*	**EVALUATIE**

In het werkcollege van 11 december zal er minstens één verplichte oefening worden gekozen waarvoor de student *individueel* zijn resultaat moet laten zien aan de begeleider. Door middel van bijkomende vragen zal worden gepeild naar het beheersen van de vaardigheden van de student. Wanneer de gekozen oefening ondermaats is, kan er eventueel op vraag van de student nog een tweede oefening geëvalueerd worden voor 2/3 van de punten.

Indienen: *Je oplossingen moeten ingediend worden vóór 11 december (dus ten laatste op 10 december om 23u59).* Zorg ervoor dat je alle oplossingen voor de evaluaties op een duidelijke manier van commentaar voorziet. Het documenteren van je oplossingen maakt ze immers begrijpbaar voor de evaluatoren én voor jezelf. Oplossingen die moeilijk begrijpbaar zijn omdat de nodige commentaar ontbreekt zullen lager gequoteerd worden.

Alle aangemaakte bestanden, moeten als volgt samengevoegd worden in een ZIP-bestand:

```
zip oplossingen.zip <lijst van aangemaakte bestanden>
```

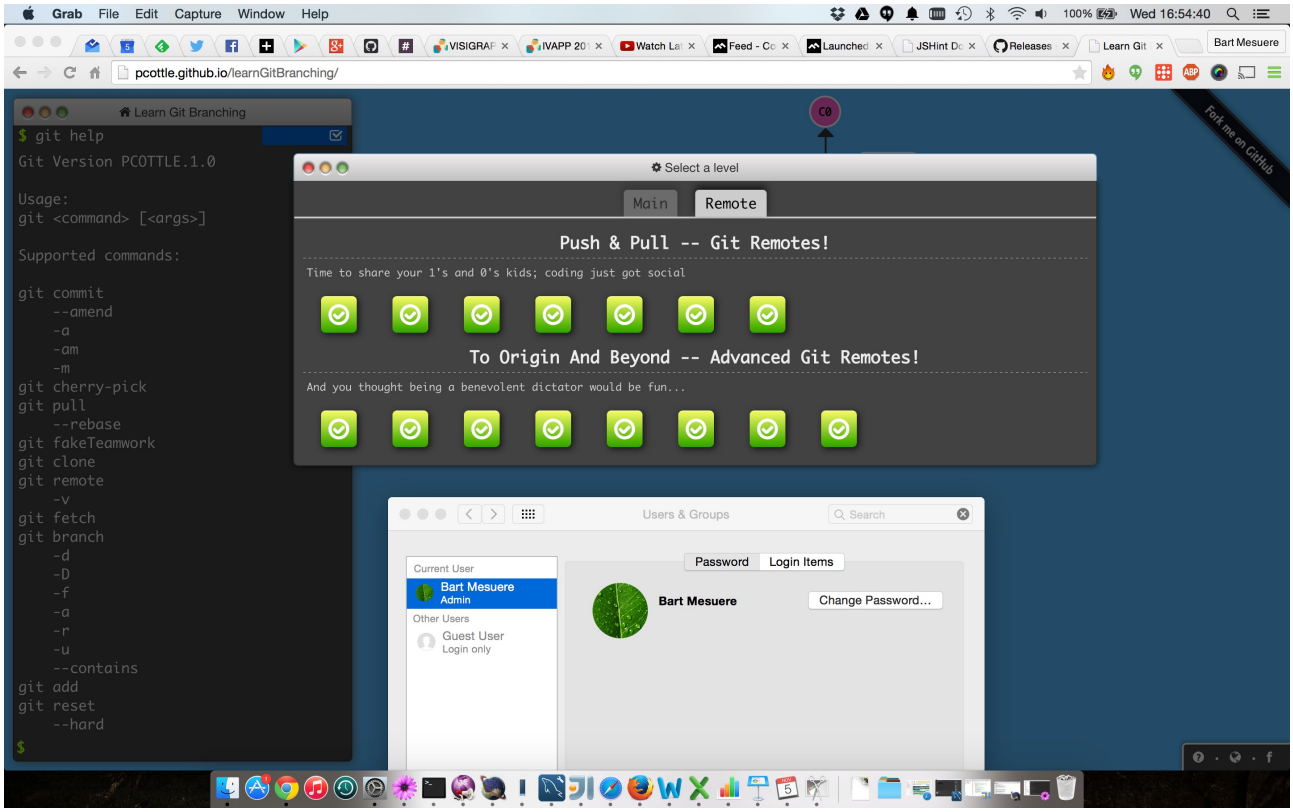
Windows gebruikers kunnen het zo bekomen bestand van helios naar hun lokale schijf kopiëren met behulp van WinSCP (`http://helpdesk.UGent.be/netdisk/bestand_scp.php`). Als je Linux gebruikt, dan kun je dit met het commando `scp` doen. Het bekomen ZIP-bestand moet vervolgens ingediend worden via de `http://indianio.UGent.be` webapplicatie. Let er hierbij op dat je de instructies nauwgezet volgt en dat je ZIP-bestand op de correcte manier werd aangemaakt!

Werkcollege 6: Git

Oefening 6.1

B ★

Om kennis te maken met de verschillende git commando's zullen we de online oefeningen van de website `http://pcottle.github.io/learnGitBranching/` gebruiken. Het is voor dit werkcollege de bedoeling om zowel de 18 *levels* van het *Main* gedeelte op te lossen, als de 15 *levels* van het *Remote* gedeelte. Hou voor elke oefening bij welke commando's je gebruikt hebt in het tekstbestand `git.txt` en neem na het afwerken van elk van de twee gedeeltes een gepersonaliseerd screenshot waarop te zien is dat je alles hebt opgelost. Een voorbeeld van dergelijk screenshot kan je hieronder bewonderen.



Werkcollege 7: Git en Github

Oefening 7.1 B ⋆

De oplossingen van de tweede oefeningenreeks zullen we bijhouden in een *git repository* in plaats van in een tekstbestand zoals bij reeks 1. Maak dus op `http://github.ugent.be` een ***private*** *repository* aan waarin je telkens alle oplossingen zult plaatsen. Als je voor de tweede oefeningenreeks samenwerkt met een collega, dan maak je 1 gezamenlijk repository aan waar je beiden alle rechten op hebt. Probeer alles wat ordelijk te houden door bijvoorbeeld voor elke oefening een apart mapje aan te maken binnen je repository. Gebruik ook een `README.md` bestand in de **root** van je repository om een overzicht van de opgeloste oefeningen bij te houden waarin je telkens linkt naar de locatie van de oplossing. Gevraagd wordt ook om de repository doorheen het semester te gebruiken; met andere woorden het is **niet** de bedoeling om vlak voor de deadline alle gemaakte oefeningen toe te voegen.

Als github je vraagt of het repository al geïnitialiseerd moet worden, dan antwoord je nee. We zullen dit namelijk handmatig doen in de volgende oefening.

Maak gerust gebruik van *branches*, *issues* en *pull requests* om ook voorlopige, half afgewerkte oplossingen toe te voegen.

Oefening 7.2 ⋆

Maak een nieuwe map aan in je helios home directory of op het unix systeem waarop je deze oefening oplost en voer onderstaande opdrachten uit:

1. Gebruik het `init` commando om een nieuwe repository te initialiseren in de net aangemaakte map.
2. Observeer of er een (verborgen) `.git` map is aangemaakt. Wat kan je hierin vinden?
3. Maak een `README`-bestand aan.
4. Gebruik het `status` commando om te kijken wat de status is van het bestand. Wat wil dit zeggen?
5. Gebruik het *add* commando om het bestand toe te voegen aan de *staging area*.
6. Voer opnieuw het `status` commando uit. Wat is het verschil?
7. Gebruik het `commit` commando om de inhoud van de staging area te *committen*.
8. Maak een map met de naam `oefening_7.2` aan en voeg enkele bestanden toe aan deze map.
9. Gebruik het `add` commando om deze bestanden toe te voegen aan de staging area, maar zonder de naam van de bestanden zelf te gebruiken. Controleer of het gelukt is met het `status` commando en commit ze uiteindelijk met het `commit` commando.
10. Pas een van de bestanden aan en gebruik het (git) `diff` commando om de wijzigingen te bekijken.

11. Gebruik `add` op het gewijzigde bestand en voer opnieuw `diff` uit. Wat krijg je te zien? Wat moet je doen om de wijzigingen te zien te krijgen? Commit alle wijzigingen.
12. Gebruik het `log` commando om een overzicht te krijgen van de afgelopen commits.
13. Gebruik het `show` commando om de details van een individuele commit te bekijken.
14. Gebruik het git `rm` commando om een bestand te verwijderen en bekijk de `status`. Commit de wijziging.
15. Gebruik het niet-git `rm` commando om een bestand te verwijderen en bekijk de `status`. Wat is het verschil met de status uit het vorige puntje? Commit de wijziging.
16. Gebruik het `diff` commando om alle wijzigingen te zien die geïntroduceerd werden in de eerste drie commits.

Oefening 7.3 ⋆

Werk verder in de repository van oefening 7.2 en voer onderstaande opdrachten uit:

1. Gebruik het `status` commando om te zien op welke branch je zit.
2. Gebruik het `branch` commando om een nieuwe branch `7.3` aan te maken.
3. Blijf op de master-branch, pas de eerste regel van het `README` bestand aan en commit de aanpassingen.
4. Kijk met `log` of je laatste commit in het lijstje staat.
5. Gebruik het `checkout` commando om naar de `7.3` branch over te schakelen.
6. Voer nogmaals `log` uit en controleer of je vorige commit daar te vinden is. Bekijk ook de eerste regel van het `README` bestand om de toestand te controleren.
7. Pas de eerste regel van het `README` bestand aan (iets anders dan in stap 3) en commit de aanpassingen.
8. Schakel terug over op de `master` branch.
9. Gebruik `merge` om de aanpassingen van branch `7.3` aan de master toe te voegen. Wat gebeurt er?
10. Open het `README` bestand en bekijk de inhoud. Wat merk je op?
11. Verhelp het probleem en bekijk de uitvoer van `status`.
12. Commit de aanpassingen.
13. Gebruik `log` om te verifiëren of alles klopt.

Oefening 7.4 ⋆

Werk verder in de repository van oefening 7.2 en voer onderstaande opdrachten uit:

1. Gebruik het `remote` commando om de locatie van de repository uit oefening 7.1 toe te voegen.
2. Gebruik `push` om alle commits op de master branch naar github te sturen.

3. Controleer op de github website of dit gelukt is.
4. Bekijk de *network graph* van je repository op github. Komt dit overeen met wat je verwacht na de vorige oefeningen op te lossen?
5. `clone` de repository op de unix-machine waarop je werkt zodat je dezelfde repository op twee verschillende plaatsen op de machine hebt staan.
6. Doe een aanpassing aan een bestand, commit de wijzigingen en `push` alles naar de server.
7. Navigeer op de machine naar de andere locatie van de repository, maak een aanpassing aan een ander bestand, commit de wijzigingen en `push` alles naar de server. Waarom lukt dit niet?
8. Verhelp het probleem zodat ook je commit op de tweede locatie op de server belandt.
9. Maak een nieuwe wijziging en commit deze.
10. Gebruik de `--amend` parameter om de vorige commit aan te passen en `push` alles naar de server.
11. Gebruik nogmaals `--amend` om de vorige commit aan te passen en probeer opnieuw alles naar de server te pushen. Wat loopt er fout en waarom? Is de `-f` vlag gebruiken een goeie oplossing?

Oefening 7.5 ⋆

Werk verder in de repository van oefening 7.2 en voer onderstaande opdrachten uit:

1. Maak een nieuwe branch `7.5` aan en wissel naar deze branch
2. Maak een nieuwe commit en push de `7.5` branch naar de server.
3. Wissel naar de master branch en maak daar een nieuwe commit.
4. Wissel opnieuw naar de `7.5` branch en gebruik `rebase` om je vorige commit op master in deze branch te krijgen.
5. `push` de 7.5 branch opnieuw naar de server. Waarom lukt dit niet? Hoe zou je dit probleem kunnen oplossen?
6. Hoe zou je zulke problemen in de toekomst kunnen voorkomen? Bedenk minstens twee verschillende oplossingen.

Oefening 7.6 ⋆

Maak op `http://github.ugent.be` een nieuwe publieke repository aan waarop je minstens de volgende acties uitvoert:

- Initialiseer de repository via de webinterface.
- Pas een bestand aan via de webinterface.
- Maak een issue aan.
- Maak een nieuwe branch aan.

- Sluit een issue vanuit een commit message.
- Voeg een tag toe.
- Kloon de repository van een collega.
- Doe een aanpassing in de gekloonde repository en open een pull request bij je collega.
- Voeg commentaar toe op een regel code in de aangemaakte pull request.

Werkcollege 8: Bash

Oefening 8.1 E ⋆

Het denkspel *Bloemblaadjes rond de Roos* wordt gespeeld met vijf dobbelstenen. Het wordt gespeeld door een computerprogramma of met echte dobbelstenen die worden geworpen door een *Potentaat van de Roos*, een persoon die het geheim achter het spel kent. Voor elke worp met de dobbelstenen is er een unieke numerieke oplossing. De spelers moeten de juiste oplossing proberen te achterhalen via inductief redeneren. Indien ze er niet in slagen om de juiste oplossing te raden, wordt de juiste oplossing meegedeeld door de *Potentaat van de Roos*, en moeten ze proberen om bij een volgende worp de juiste oplossing te raden.

Het spel kent slechts drie regels:

1. De naam van het spel is *Bloemblaadjes rond de Roos*, en deze naam is belangrijk.
2. Het antwoord is steeds een niet-negatief even geheel getal.
3. Iedereen die het geheim van het spel kent, mag de juiste oplossing geven die correspondeert met een bepaalde worp, maar mag het geheim zelf nooit prijsgeven.

Gevraagd wordt om een bash shell script `bloemblaadjes.sh` te schrijven, dat de rol van *Potentaat van de Roos* op zich neemt. Om het geheim van het spel niet te onthullen, wordt de oplossing in dit shell script niet rechtstreeks berekend. In plaats daarvan wordt gebruik gemaakt van een bestand `.bloemblaadjes` dat zich in de home directory van de speler bevindt. Dit bestand bevat een lijst van oplossingen, elk op een afzonderlijke regel die zes gehele getallen bevat, van elkaar gescheiden door een spatie. De eerste vijf getallen stellen een worp met vijf dobbelstenen voor, terwijl het zesde getal de corresponderende oplossing voorstelt. Het bestand kan gedownload worden vanaf `http://computergebruik.ugent.be/oefeningenreeks2/bloemblaadjes.txt`.

Voor het simuleren van elke worp moet het shell script een willekeurige regel uit `.bloemblaadjes` selecteren. Van deze regel worden de eerste vijf getallen naar het beeldscherm geschreven, waarna het script aan de speler vraagt om de corresponderende oplossing te geven. Indien de speler de waarde `E` of e geeft, wordt het script afgesloten. In het andere geval bepaalt het script of het om een correcte oplossing gaat.

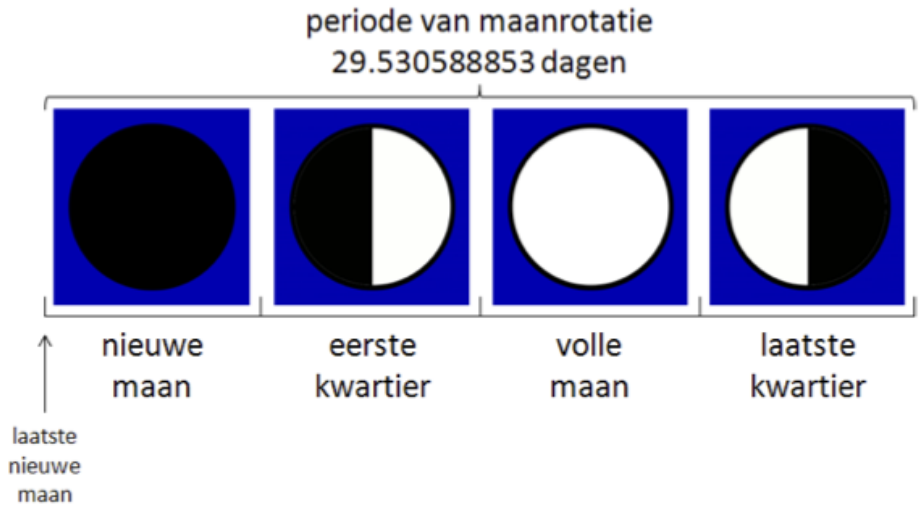
Bij wijze van illustratie vind je hieronder een voorbeeldsessie van het spel Bloemblaadjes rond de Roos zoals het door het shell script moet kunnen gespeeld worden.

```
1 $ ./bloemblaadjes.sh
2 Worp: 5 4 4 2 2
3 Hoeveel bloemblaadjes staan er rond de roos? 2
4 Fout! Er staan 4 bloemblaadjes rond de roos.
5 $ ./bloemblaadjes.sh
6 Worp: 5 1 5 5 5
7 Hoeveel bloemblaadjes staan er rond de roos? 16
8 Correct! Er staan 16 bloemblaadjes rond de roos!
9 (bij deze ben je gebonden aan strikte geheimhouding)
```

Tip: Voor het inlezen van gebruikersinvoer kan je het `read` commando gebruiken.

Oefening 8.2 E ⋆

Onze maan vertoont zogenaamde *schijngestalten*, wat wil zeggen dat ze zich gedurende haar omloop rond de aarde in verschillende gedaanten laat zien. Dit fenomeen wordt veroorzaakt door het feit dat de maan vanuit een andere richting door de zon wordt bestraald dan de richting waarin ze wordt waargenomen. Hierdoor lijkt het alsof de *terminator* (dit is de scheidingslijn van licht en donker) zich over de oppervlakte van de maan verplaatst, waardoor de maan te zien is als een sikkelvormig of rond lichaam. De periode waarin de maan rond de aarde draait wordt over het algemeen onderverdeeld in vier *maanfasen*: nieuwe maan, eerste kwartier, volle maan en laatste kwartier. Als je voor een gegeven datum kunt bepalen hoeveel dagen er zijn verstreken sinds de laatste nieuwe maan, dan kan je de corresponderende maanfase bepalen door te kijken in welke vierde deel van de periode van de maanrotatie dit aantal dagen valt (zie onderstaande figuur).



De maan draait in een baan om de aarde met een periode van $p = 29.530588853$ dagen. Als we weten dat er een nieuwe maan voorkwam op 6 januari 2000, dan kunnen we op basis van onderstaande formule het aantal dagen sinds de laatste nieuwe maan bepalen. We noteren dit aantal dagen als $d_n \in \mathbb{R}$.

$$d_n = d_r - p * \frac{d_r}{p}$$

Hierbij stelt d_r het aantal dagen voor sinds de referentiedatum van 6 januari 2000, en stelt de breuk een gehele deling voor (of met andere woorden $d_n = d_r \bmod p$, waarbij $p \in \mathbb{R}$). Stel dat we bijvoorbeeld de maanfase willen bepalen voor de datum 21/06/2010. Er zijn op deze datum reeds 3819 dagen verstreken sinds 6 januari 2000, waardoor in dit geval $d_n = 9.55$. Aangezien de waarde van d_n in het tweede deel van de vierdelige onderverdeling van de maanperiode valt, correspondeert deze datum dus met de maanfase *eerste kwartier*.

Gevraagd wordt om een `bash` shell script `maanfase.sh` te schrijven. Aan het shell script moeten twee gehele getallen als argument meegegeven worden, die respectievelijk het volgnummer van een maand en een jaartal voorstellen. Het shell script moet n enkele regel teruggeven waarop de maanfasen voor de verschillende dagen van de maand zijn weergegeven. Hierbij mag verondersteld worden dat elke maand 31 dagen telt. Voorts wordt de maanfase *nieuwe maan* voorgesteld door `@`, *eerste kwartier* door `)`, *volle maan* door `O` en *laatste kwartier* door `(`.

Onderstaande sessie illustreert de werking van het shell script `maanfase.sh`.

```
$ ./maanfase.sh 6 2010
1O2O3O4O5(6(7(8(9(10(11(12@13@14@15@16@17@18@19)20)21)22)23)24)25)26)27O28O29O30O31O
$ ./maanfase.sh 6 2009
1)2)3)4)5)6)7)8O9O10O11O12O13O14O15(16(17(18(19(20(21(22(23@24@25@26@27@28@29@30)31)
```

Hierbij geeft de uitvoer `1O2O3O4O5(6(...` bijvoorbeeld aan dat er een volle maan voorkomt op de eerste dag van de maand, en dat de maan zich in het laatste kwartier bevindt op de zesde dag van de maand.

Om te berekenen hoeveel dagen zijn verstreken sinds 6 januari 2000, kan het Juliaanse dagnummer gebruikt worden. Voor een gegeven datum `dd/mm/jjjj` kan dit als volgt berekend worden:

$a \leftarrow (14 - \texttt{mm})/12$
$y \leftarrow \texttt{jjjj} + 4800 - a$
$m \leftarrow 12 * a - 3 + \texttt{mm}$
$jdn \leftarrow \texttt{dd} + (153 * m + 2)/5 + 365 * y + y/4 - y/100 + y/400 - 32045$

Alle delingen stellen hierbij gehele delingen voor. Het nummer voor de datum `21/06/2010` is bijvoorbeeld `2455369`.

Tip: Als je gebruik maakt van het commando `bc` om te rekenen met reële getallen, dan zal je merken dat het commando `echo "4/3"|bc` als resultaat de waarde `1` teruggeeft. De waarde die je toekent aan de `bc`-variabele `scale` geeft aan hoeveel cijfers na de komma het resultaat moet bevatten. Deze waarde moet toegekend worden voor het uitvoeren van de berekening en is standaard ingesteld op nul. Het commando `echo "scale=2;4/3"|bc` zal dus de waarde `1.33` teruggeven.

Tip: De GNU versie van het commando `bc` kan ook booleaanse expressies evalueren. Het commando `echo "7>3|bc"` geeft de waarde `1` terug, en het commando `echo "7<3|bc"` geeft de waarde `0` terug.

Werkcollege 9: Bash

Oefening 9.1 E ⋆

Beschouw een string die elk van de 26 letters van het alfabet juist één keer bevat, bijvoorbeeld

```
yhcwslaedmriopvgzxkbjntuqf
```

Deze permutatie kan gebruikt worden als de sleutel van een substitutiecodering, die in het geval van bovenstaand voorbeeld de letter `a` afbeeldt op de letter `y`, de letter `b` op de letter `h`, enzoverder. Deze sleutel codeert bijgevolg het woord `punker` als `gjprsx`. Merk hierbij op dat de letters van het gecodeerde woord `gjprsx` in alfabetische volgorde staan. De uitdaging bestaat erin om een permutatie van de letters van het alfabet te vinden, die een maximaal aantal woorden uit een gegeven woordenlijst codeert als woorden die in alfabetische volgorde staan.

Schrijf een `bash` shell script `evalueerSleutel.sh` dat kan gebruikt worden om de score te bepalen van het bovenstaande optimalisatieprobleem voor een gegeven sleutel en een gegeven woordenlijst. De score wordt bepaald als het aantal woorden van de gegeven woordenlijst dat gecodeerd wordt als woord waarvan de letters in alfabetische volgorde staan. Deze score moet door het script naar standaard uitvoer geschreven worden.

Aan het script moet verplicht een sleutel als stringargument doorgegeven worden, die een permutatie van de letters van het alfabet voorstelt. Om de woordenlijst op te halen waarop het optimalisatieprobleem moet beoordeeld worden, kijkt het script achtereenvolgens naar onderstaande invoerkanalen. Het eerste invoerkanaal dat gevonden wordt, wordt door het script gebruikt.

1. een bestand met bestandsnaam die als tweede optionele argument aan het script wordt doorgegeven
2. een bestand met bestandsnaam in de omgevingsvariabele `INVOER`
3. standaard invoer

In volgende gevallen moet het script een gepaste *exit status* instellen, en een foutboodschap uitschrijven naar *standard error* zoals aangegeven in onderstaande voorbeeldsessie:

- aan het script worden niet één of twee argumenten doorgegeven (*exit status* 1)
- de doorgegeven sleutel vormt geen permutatie van het alfabet (*exit status* 2)
- het script moet zelf een bestand openen waaruit de woordenlijst moet gelezen worden, en dit bestand kan niet gelezen worden (*exit status* 3)

```
1  $ evalueerSleutel.sh jhcwslaedmviotrgzxkbynpuqf 6-letter-woorden.txt
2  23
3  $ evalueerSleutel.sh kecxuarhobtpdqifgwysmljvzn < 6-letter-woorden.txt
4  379
5  $ export INVOER="6-letter-woorden.txt"
6  $ evalueerSleutel.sh idsxvaqtobuefpgcjwzrkmhnyl
7  474
8
9  $ evalueerSleutel.sh
10 gebruik: evalueer_sleutel.sh sleutel [bestand]
11 $ evalueerSleutel.sh x y z
12 gebruik: evalueer_sleutel.sh sleutel [bestand]
```

```
13
14 $ evalueerSleutel.sh abc
15 fout: sleutel is geen permutatie van het alfabet
16 $ evalueerSleutel.sh xxxwvutsrqponmlkjihgfedcba
17 fout: sleutel is geen permutatie van het alfabet
18
19 $ evalueerSleutel.sh kecxuarhobtpdqifgwysmljvzn 999-letter-woorden.txt
20 fout: bestand 999-letter-woorden.txt kan niet gelezen worden
```

Het voorbeeldbestand kan gedownload worden vanaf `http://computergebruik.ugent.be/oefeningenreeks2/6-letter-woorden.txt`.

Oefening 9.2 E ⋆

De term *magic number* wordt gebruikt voor een constante bit- of bytesequentie die altijd op dezelfde positie voorkomt binnen bestanden van een bepaald bestandsformaat. Deze magic numbers kunnen daardoor gebruikt worden om het bestandsformaat van een gegeven bestand te achterhalen. Dit is het principe waarop het `file` commando in Unix gebaseerd is.

Gevraagd wordt om een `bash` shell script `findmagic.sh` te schrijven, dat kan gebruikt worden om de magic numbers van een bestandsformaat te achterhalen. Aan dit script moeten minstens twee gewone bestanden als argument meegegeven worden. Het script moet de lijst van byteposities die identiek zijn voor alle gegeven bestanden naar standaard uitvoer schrijven. Elke regel van deze lijst bestaat uit het volgnummer van een identieke byte (geïndexeerd vanaf 0; offset ten opzichte van het begin van het bestand), gevolgd door een dubbelpunt, een spatie en de waarde van de byte zelf. Afdrukbare karakters moeten als dusdanig uitgeschreven worden, terwijl niet-afdrukbare karakters (ASCII-waarde kleiner dan 32 of gelijk aan 127) in hexadecimale vorm moeten uitgeschreven worden: `\0x`*`HH`* (waarbij *`HH`* twee hexadecimale cijfers zijn). Onderzoek telkens de eerste 100 bytes van de bestanden. Onderstaande sessie illustreert de werking van het shell script `findmagic.sh`. Hieruit kan afgeleid worden dat men PDF bestanden kan herkennen aan het feit dat de eerste 5 bytes van dergelijke bestanden de tekst `%PDF-` bevatten.

```
1  $ findmagic.sh *.pdf
2  0: %
3  1: P
4  2: D
5  3: F
6  4: -
7  5: 1
8  6: .
9  9: %
10 $ findmagic *.jpg
11 0: \0xFF
12 1: \0xD8
13 2: \0xFF
14 3: \0xE0
15 4: \0x0A
16 5: \0x10
17 6: J
18 7: F
19 8: I
20 9: F
21 ...
```

Het shell script `findmagic.sh` moet de geldigheid van de argumenten nagaan, namelijk minstens twee argumenten en uitsluitend bestaande gewone bestanden. Het shell script moet een gepaste foutboodschap uitschrijven indien niet aan deze voorwaarden voldaan wordt. Dit wordt geïllustreerd in onderstaande sessie.

```
$ findmagic.sh test1.pdf
findmagic.sh: at least two regular files should be passed as an argument.
Usage: findmagic.sh file file ...
$ findmagic.sh test1.pdf test2.pdf test3.pdf /etc test4.pdf
findmagic.sh: /etc is not a regular file.
Usage: findmagic.sh file file ...
```

Tip: Om een byte op een bepaalde positie in een bestand uit te lezen, en om deze byte om te zetten naar zijn decimale of hexadecimale ASCII-waarde, kunnen de volgende commando's gebruikt worden.

```
# byte op positie p (offset tov begin) in gegeven bestand uitschrijven
dd if=<bestand> bs=1 count=1 skip=<p> 2> /dev/null

# decimale ASCII-waarde van byte "A" uitschrijven
echo "A" | od -td1 | awk 'NR==1{print $2}'

# hexadecimale ASCII-waarde van byte "A" uitschrijven
echo "A" | xxd -u -l1 | awk '{print $2}'
```

Werkcollege 10: Bash

Oefening 10.1 E ⋆

Schrijf een `bash` shell script `diffdir.sh` dat kan gebruikt worden om twee directories met elkaar te vergelijken. Aan dit shell script moeten twee padnamen van directories als argument doorgegeven worden. Indien de doorgegeven argumenten niet correct zijn, moet een gepaste foutboodschap uitgeschreven worden naar *standard error* en moet een `exit`-waarde 1 teruggegeven worden (zie onderstaande voorbeeldsessie).

```
1  $ diffdir.sh
2  Gebruik: diffdir DIRECTORY DIRECTORY
3  $ echo $?
4  1
5  $ diffdir.sh orig
6  Gebruik: diffdir DIRECTORY DIRECTORY
7  $ diffdir.sh orig diffdir.sh
8  Gebruik: diffdir DIRECTORY DIRECTORY
9  $ diffdir.sh diffdir.sh copy
10 Gebruik: diffdir DIRECTORY DIRECTORY
```

Bij het vergelijken van de directories moet het shell script `diffdir.sh` de bestanden oplijsten die verwijderd, toegevoegd en verplaatst werden (waarbij de eerste directory als referentiepunt gebruikt wordt). Bestanden worden hierbij niet geïdentificeerd door hun naam of padnaam, maar door een SHA-digest die berekend wordt op basis van de inhoud van de bestanden. De SHA-digest is een hashwaarde die kan bepaald worden aan de hand van het commando `shasum`, en die dus niet wijzigt als het bestand hernoemd, verplaatst of gekopieerd wordt.

```
1  $ shasum oude_naam
2  3add07924b6c05ab09a64a66b02a389df24a896c  oude_naam
3  $ mv oude_naam nieuwe_naam
4  $ shasum nieuwe_naam
5  3add07924b6c05ab09a64a66b02a389df24a896c  nieuwe_naam
```

Het shell script `diffdir.sh` moet dus eerst de SHA-digests bepalen van alle gewone bestanden (andere bestandstypes moeten niet meegenomen worden) uit de twee gegeven directories (inclusief hun subdirectories). Op basis van een vergelijking van de twee lijsten van SHA-digests (hiervoor kan bijvoorbeeld het commando `comm` gebruikt worden) moet het shell script de volgende categorien van bestanden oplijsten:

- verwijderde bestanden: bestanden die voorkomen in de eerste directory, maar niet in de tweede
- toegevoegde bestanden: bestanden die voorkomen in de tweede directory, maar niet in de eerste
- verplaatste bestanden: bestanden die voorkomen in beide directories, maar op een andere locatie relatief ten opzichte van de padnaam van de directory

Indien er bij de vergelijking geen bestanden gevonden werden voor een bepaalde categorie, dan moet er voor die categorie niets uitgeschreven worden. Anders moet eerst een regel met het type van de categorie uitgeschreven worden, gevolgd door een oplijsting van de relatieve padnamen van alle bestanden uit die categorie (relatief ten opzichte van de opgegeven padnamen van de directories die vergeleken worden). Voor de verplaatste bestanden moet eerst de padnaam in de eerste directory vermeld worden, gevolgd door een groter dan teken (`>`) en de padnaam in de tweede directory. De padnamen moeten alfabetisch uitgeschreven worden per categorie, elk

op een afzonderlijke regel en telkens voorafgegaan door twee spaties. Dit wordt geïllustreerd door onderstaande voorbeeldsessie.

```
$ diffdir.sh orig copy
verwijderde bestanden:
  GHI/jdjs.cpp
  lzdj.tex
toegevoegde bestanden:
  DEF/iejd.zip
  eksf.gif
verplaatste bestanden:
  DEF/slds.sh > GHI/slds.sh
  GHI/eejd.py > DEF/lfkf.pl
  jhrd.png > DEF/dsle.cpp
$ diffdir.sh orig/DEF copy/GHI
verwijderde bestanden:
  ksod.png
  kzjr.tex
toegevoegde bestanden:
  ekje.tex
  opzd.zip
```

Je mag er bij het schrijven van het shell script `diffdir.sh` van uitgaan dat alle bestanden uit een directory (die als argument aan het script wordt doorgegeven) een verschillende inhoud hebben. Dit betekent niet strikt noodzakelijk dat geen twee bestanden uit eenzelfde directory dezelfde SHA-digest hebben, maar ook daar mag je van uitgaan. Indien je voor de werking van het shell script `diffdir.sh` tijdelijke bestanden nodig hebt, probeer deze dan op een zo veilig mogelijke manier aan te maken, en zorg er zeker voor dat ze vóór het beëindigen van het script terug verwijderd worden. Merk ook op dat als enkel de inhoud van een bestand werd bijgewerkt zonder dat het bestand werd verplaatst of hernoemd, dit volgens de definitie van het shell script `diffdir.sh` wordt gezien als een bestand dat werd verwijderd uit de eerste directory en een nieuw bestand (met zelfde relatieve padnaam) dat werd toegevoegd aan de tweede directory.

De voorbeeldbestanden kunnen gedownload worden vanaf `http://computergebruik.ugent.be/oefeningenreeks2/diffdirvoorbeeld.zip`.

Oefening 10.2 **E** ⋆

Machinevertalingen zoals die van Google Translate zijn nog steeds niet perfect. Als we bijvoorbeeld een stuk Nederlandse tekst automatisch naar het Engels laten vertalen, en het resultaat vervolgens opnieuw naar het Nederlands laten vertalen, dan is de kans erg klein dat we opnieuw de originele tekst uitkomen.

Schrijf een `bash` shell script `vertaling.sh` dat als invoer twee talen en een zin in de eerste taal meekrijgt, en deze zin blijft vertalen tussen de twee opgegeven talen totdat de vertaling stabiel is. Dit wil zeggen, totdat de zin na een heen- en weervertaling niet meer verandert.

Invoer

Het script moet opgeroepen kunnen worden met de volgende opties en argumenten:

- `-v`: optionele optie die de *verbose* modus inschakelt
- `-c <cycli>`: optionele optie die via het bijhorende argument aangeeft hoeveel cycli er maximaal moeten uitgevoerd worden; als deze optie niet wordt meegegeven, dan worden standaard maximaal 10 cycli uitgevoerd

- `-i <brontaal>`: verplichte optie die via het bijhorende argument aangeeft wat de taal van de opgeven zin is (bv. *nl*, *en*, *fr*, ...)
- `-o <doeltaal>`: verplichte optie die via het bijhorende argument aangeeft naar welke taal de zin moet vertaald worden
- de overgebleven argumenten moeten als de te vertalen tekst genterpreteerd worden

Bij verkeerd gebruik van de opties (bijvoorbeeld het gebruik van een optie die niet ondersteund wordt, het niet opgeven van een verplichte optie of het niet opgeven van een verplicht argument bij een optie) moet het shell script een gepaste foutmelding uitschrijven naar `stderr`, die aangeeft hoe het script op een correcte manier moet gebruikt worden.

Uitvoer

Het script moet de gegeven zin vertalen vanuit de brontaal naar de doeltaal. Het resultaat van deze vertaling moet vervolgens terug vertaald worden vanuit de doeltaal naar de brontaal. Deze vertaalcyclus moet herhaald worden totdat de zin in de brontaal aan het begin van een cyclus gelijk is aan deze die op het einde van de cyclus bekomen wordt, of todat het opgegeven maximaal aantal vertaalcycli werd uitgevoerd.

Standaard wordt enkel de opgegeven zin uitgeschreven en het resultaat van de vertaling na de laatste cyclus. Indien de *verbose* modus is ingeschakeld, dan moeten alle tussenliggende vertalingen ook uitgeschreven worden, zowel in de brontaal als in de doeltaal.

Vertalingen

Voor het vertalen van de zinnen moet het shell script gebruik maken van de Google Translate API. Hiervoor kunnen de volgende commando's gebruikt worden, waarbij `${brontaal}` een variabele is die de brontaal aangeeft, `${doeltaal}` een variabele die de doeltaal aangeeft, en `${zin}` een variabele die de te vertalen zin aangeeft. De vertaling wordt toegekend aan de variabele `${vertaling}`.

```
1 antwoord=$(curl -s -i --user-agent "" -d "sl=${brontaal}" -d "tl=${doeltaal}" --data-urlencode
      "text=${zin}" https://translate.google.com)
2 vertaling=$(echo ${antwoord} | sed "s/^.*id=result_box[^<]*<span[^>]*>\([^<]*\)<\/span>.*$
     /\1/" | sed "s/&[^;]*;//g")
```

Voorbeeld

```
1  $ vertaling.sh -i nl -o en "Dit is een moeilijke zin"
2  Dit is een moeilijke zin
3  Dit is een moeilijke frase
4
5  $ vertaling.sh -vi nl -o en "Dit is een moeilijke zin"
6  Dit is een moeilijke zin
7  This is a difficult phrase
8  Dit is een moeilijke frase
9  This is a difficult phrase
10 Dit is een moeilijke frase
```