

Besturingssystemen 2

Bert De Saffel

16 juni 2017

Inhoudsopgave

I	Unix Theorie	3
1	Basic commando's en direcorey hiërarchie	4
1.1	Inleiding	4
1.2	De shell	4
1.3	Shell variabelen	5
1.4	Shell instellingen	5
1.5	Interne en externe functies	5
1.6	I/O	5
1.7	Help en manpages	5
1.8	Directories	6
1.9	Symbolische links	6
1.10	Shell Globbing	7
1.11	Basic Commands	7
1.11.1	ls	7
1.11.2	file	8
1.11.3	stat	8
1.11.4	cp	8
1.11.5	mv	8
1.11.6	rm	9
1.11.7	touch	9
1.11.8	comm	9
1.11.9	diff	9
1.11.10	Message Digest	10
1.11.11	paste	10
1.11.12	join	10
1.11.13	split	10
1.11.14	echo	11
1.11.15	printf	11
1.11.16	locate	11
1.11.17	find	11
1.12	Intermediate Commands	12
1.12.1	cat	12
1.12.2	tac	13

1.12.3	rev	13
1.12.4	wc	13
1.12.5	head	13
1.12.6	tail	13
1.12.7	nl	14
1.12.8	cut	14
1.12.9	strings	14
1.12.10	sort	14
2	Devices	16
2.1	Devices Files	16
2.1.1	Block devices	16
2.1.2	Character devices	16
2.2	Terminals	19
3	Shell Scripts	20
3.1	Shell Script Basics	20
3.1.1	2. Brace Expansion	20
3.1.2	5. Command Substitution	21
3.1.3	6. Process Substitution	21
3.1.4	? Arithmetic Substitution	21
3.1.5	? Parameter Expansion	21
3.2	Functies	22
3.3	Variabelen	23
3.3.1	read	23
3.4	Foutcodes	24
3.5	Arrays	24
3.5.1	Numeriek	24
3.5.2	Associatief	24
3.6	Conditionele logica	24
3.6.1	If tests	24
3.6.2	Case	25
3.7	For loops	25
3.8	While loops	25
3.8.1	Teken per teken verwerken	26
3.8.2	Verwerken uitvoer opdracht	26
3.9	Tijdelijke bestanden	26
3.10	Here Documents en Strings	26
3.11	Signalisering	26

Deel I

Unix Theorie

Hoofdstuk 1

Basic commando's en direcory hiërarchie

1.1 Inleiding

Er bestaan verschillende Unix varianten die elk hun eigen manier hanteren om de user te laten communiceren met de kernel en wederom. Het enige dat ze zeker gemeenschappelijk hebben is het feit dat ze de POSIX standaard volgen. Hierdoor ontstaat er compabiliteit tussen de verschillende varianten. Tegenwoordig worden de verschillende Unix varianten geleverd als een hele distributie. Deze distributies bevatten typisch:

- De kernel (systeemaanroepen en POSIX)
- Services (servers, daemons)
- Utilities (GNU)
- Programmeromgeving
- GUI omgeving

1.2 De shell

De shell is een C(ommand) L(ine) I(nterface) waardoor er gebruik kan gemaakt worden van de verschillende functies die het besturingssysteem beschikbaar stelt. Het is een command interpreter en kan dus shell scripts uitvoeren. Er zijn verschillende soorten Unix shells beschikbaar zoals: Bourne (.sh), Korn (.ksh), Joy (.csh) en Falstad (.zsh). Om te weten welke shell er op een bepaald systeem staat wordt het commando *echo \$SHELL*. In de cursus wordt de Bourne shell(bash) gebruikt.

1.3 Shell variabelen

Het is mogelijk om globale variabelen in te stellen die kunnen gebruikt worden in het huidige proces en eventueel onderliggende kindprocessen. Om een lijst op te vragen van alle variabelen wordt er gebruik gemaakt van het commando *declare -p*. Om een variabele in te stellen wordt het commando *declare x = 5874* gebruikt. Deze variabele zal alleen kunnen gebruikt worden in het huidige proces. Om een variabele globaal te definiëren zodat het ook door kindprocessen gebruikt kan worden, wordt de optie *-x* meegegeven aan het *declare* commando.

1.4 Shell instellingen

Er zijn een aantal configuratiemogelijkheden beschikbaar voor de shell als voor bash. Voor configuratie in de shell wordt er gebruik gemaakt van het commando *set*. Hier wordt er altijd een optie *±o* meegegeven. *+* zal een optie aanzetten terwijl *-* een optie zal uitschakelen. Verder zijn er ook configuratiemogelijkheden voor bash. Deze worden bewerkt via het commando *shopt*. Voor een configuratie aan te zetten wordt de optie *-s* gebruikt en voor het uit te schakelen wordt de optie *-u* gebruikt.

Belangrijke *shopt* opties zijn *extglob* en *globstar*. Belangrijke *set* opties zijn *posix*, *verbose*, *xtrace*, *noclobber*, *errexit*, *nounsout*.

1.5 Interne en externe functies

Een commando is een interne (builtin) of een externe functie. Om dit te achterhalen

1.6 I/O

1.7 Help en manpages

Unix bevat veel soorten informatie zoals *man* voor externe functies en *help* voor interne functies. Manpages worden gebruikt door het *man* commando met daarna de naam van het commando waarover meer informatie is gewenst zoals *man kill*. In het geval van *kill* is dit ook een systeemaanroep. Om alle verschillende versies van een commando te achterhalen wordt de *-f* optie gebruikt. *man -f kill* zal alle manpages van *kill* tonen. Om een specifieke manpage te openen wordt gebruik gemaakt van het commando *man x kill* waar *x* de sectienummer is. *man -f kill* is het equivalent van *what is kill*. *man -k kill* is het equivalent van *apropos kill*. *apropos* zal niet letterlijk alle *kill* commando's tonen, maar ook commando's waar *kill* in voorkomt zoals *pkill*. Een ander alternatief is het *info* commando. Dit is een vernieuwende versie van *man* en toont ook vaker voorbeelden. Tot slot hebben sommige functies ook een *—help* parameter dat

het gebruik van dit commando uitlegt. Een andere mogelijkheid is het *info* commando. Dit is een vernieuwende versie van *man* en biedt vaak ook voorbeelden. Het *man* commando wordt dus gebruikt voor informatie van externe functies. Voor interne functies moet het commando *help* gebruikt worden. Voorbeelden hiervan zijn *help shopt* en *help for*.

1.8 Directories

Om een directorystructuur weer te geven kan het commando *tree* gebruikt worden. *tree /etc* zal alle (sub)directories en bestanden die zich in */etc* bevinden uitprinten. De optie *-d* zal alleen de directories tonen, de optie *-i* zal de indentatie weglaten en de optie *-f* zal altijd het volledige pad uitprinten.

Soms komt het voor dat je tijdelijk naar een andere directory wil gaan en dan terug naar de oude directory wil gaan. Dit kan met het *pushd* en *popd* commando. *pushd* zal een directory op de stack plaatsen, alsook het als huidige locatie instellen in de shell. Om een lijst van directories op deze stack te zien wordt het commando *dirs* gebruikt. Om een directory van deze stack te halen wordt het commando *popd* gebruikt.

Een directory aanmaken kan met *mkdir* en een directory verwijderen kan met *rmdir*. Beide commando's ondersteunen de *-p* optie. Deze optie zal voor *mkdir* parent directories aanmaken indien dit nodig zou zijn en voor *rmdir* zal het de ouder directories verwijderen. *rmdir -p a/b/c* is hetzelfde als *rmdir /a/b/c a/b a*. Verder is bestaat er ook nog *rm*. Dit commando kan zowel bestanden als directories verwijderen die als parameter worden opgegeven. Standaard zal *rm* vragen voor user input bij het verwijderen van een bestand. Als je de optie *-r* meegeeft zal hij recursief de opgegeven parameter verwijderen. Hier zal er ook altijd naar input van de gebruiker gevraagd worden om al dan niet een bestand of een directory te verwijderen. Als je de *-f* meegeeft zal *rm* niet meer naar user input vragen.

1.9 Symbolische links

Een symbolische link is een verwijzing naar een bestand of een directory, al dan niet op een ander filesysteem. In tegenstelling tot een hard link zal een symbolische link verdwijnen als het origineel bestand ook verdwijnt. Het commando *ln* verwacht twee argumenten. Het eerste argument is een bestand of directory waarnaar verwezen moet worden. Het tweede argument is de naam van de link. Standaard is dit een hardlink, als je wilt dat het een symbolische link is moet je de optie *-s* meegeven. Een symbolische link verwijderen kan met het *rm* commando, maar dan moet je wel de optie *-f* meegeven. Het commando *pwd* met de optie *-P* zal de huidige directory uitprinten zonder symbolische links. Als er geen opties aan *pwd* meegegeven worden zal *-P* automatisch uitgevoerd worden. Ook het commando *cd* heeft een optie dat invloed heeft op symbolische links. Het argument *..* bij het commando *cd ..* is eigenlijk een symbolische link naar

de bovenliggende directory. Als dit niet de bedoeling is kan je aan *cd* de optie **-P** meegeven. Op die manier is .. letterlijk twee punten. Tot slot is er nog één optie bij *rm*. Als je de optie **—one-file-system** meegeeft aan *rm -rf*, zal dit enkel bestanden en directories verwijderen die op hetzelfde filesysteem zijn dan de shell waarop het commando uitgevoerd wordt.

1.10 Shell Globbing

Shell Globbing is een set van karakters dan kunnen instaan als Wildcards. Normale globbing karakters zijn: `?,*,[...],[..]en[!...]`. Er bestaan ook *character classes*: `[:space:], [:punct:], [:lower:], [:upper:], [:digit:], [:alnum:], [:alpha:]` en `[:xdigit:]`. Deze globbing kan uitgebreid worden door *extended globbing* aan te zetten. Dit wordt bereikt via *shopt -s extglob*. Nu krijg je syntax dat je aan perl doet denken. `*(expression)`, `+(expression)`, `?(expression)`, `@(expression1 — expression2 — ...)` en `!(exp)`. `*`, `+` en `?` stellen respectievelijk 0 of meer matches, 1 of meer matches en 0 of 1 match. Het `@` teken start een logische OR sequentie en `!` is een negatie.

1.11 Basic Commands

Dit onderdeel bespreekt handige commando's met hun bijhorende opties.

1.11.1 ls

`ls [OPTION]... [FILE]...`

Het *ls* commando wordt gebruikt om de inhoud van een directory te bekijken, bv *ls /etc*. Opties van *ls* zijn:

- **-l** | Gedetailleerde informatie
- **-a** | Ook bestanden die starten met een . (in feite zijn dit alle bestanden)
- **-A** | Dit toont alle bestanden behalve degene die beginnen met een . of ..
- **-d** | Zal enkel de namen van de directories printen
- **-t** | Sorteren op meest recent
- **-S** | Sorteren op grootte
- **-r** | Sorteervolgorde omkeren
- **-R** | Subdirectories recursief oplijsten
- **-1** | Zal de output niet in kolommen tonen. Dit heeft enkel effect als **-l** niet gebruikt wordt

1.11.2 file

```
file [-bchikLNNprsvz0] [--apple] [--mime-encoding] [--mime-type] [-e testname]
file -C [-m magicfiles]
file [--help]
```

Dit commando wordt gebruikt om het type van bestand te achterhalen. Zo zal *file /bin/bash* zeggen dat het een executable is, dynamically linked enz...

1.11.3 stat

```
stat [OPTION]... FILE...
```

Dit commando toont informatie zoals grootte, Inode ID, laatst gewijzigd enz... De optie **-c** is een formaatstring. In de manpagina kan je bv zien dat

1.11.4 cp

```
cp [OPTION]... [-T] SOURCE DEST
cp [OPTION]... SOURCE... DIRECTORY
cp [OPTION]... -t DIRECTORY SOURCE...
```

Dit commando wordt gebruikt om bestanden of directories te kopiëren. Het commando *cp /etc/passwd output-file* zal de inhoud van *etc/passwd* kopiëren naar *output-file*. Opties van *cp* zijn:

- **-i** | Interactieve mode. Als er een bestand zou overschreven worden zal de prompt eerst aan de user vragen voor input (y of n)
- **-f** | Negeert foutmeldingen en zal geen prompt meer vragen (gelijkaardig aan *rm -f*)
- **-u** | Zal alleen kopiëren indien het bronbestand NIEUWER is dan het doelbestand
- **-v** | Verbose modus. Zal extra debugging informatie uitprinten
- **-p** | Zal attributen niet overschrijven
- **-r** | Directories worden recursief gekopieerd
- **-a** | Combinatie van *-p* en *-r*

1.11.5 mv

```
mv [OPTION]... [-T] SOURCE
mv [OPTION]... SOURCE...
mv [OPTION]... -t DIRECTORY SOURCE...
```

Dit commando zal bestanden verplaatsen, maar dit heeft als bijgevolg dat het ook mogelijk is om bestanden te hernoemen. Het commando *mv output-file renamed-output-file* zal het bestand met als naam *output-file* hernoemen naar

renamed-output-file. Het is ook mogelijk om *mv output-file dirA/subdirB/renamed-output-file* uit te voeren. De directories worden automatisch aangemaakt indien deze niet bestaan. *mv* kent dezelfde **-i**, **-f**, **-u** en **-v** opties met dezelfde functionaliteit als *cp*.

1.11.6 rm

`rm [OPTION]... [FILE]...`

Dit commando verwijderd bestanden of directories. Het commando *rm* kent dezelfde **-i**, **-f** en **-v** opties met dezelfde functionaliteit als *cp*. Verder komt er nog de **-r** optie bij. Deze zal recursief alle subdirectories verwijderen.

1.11.7 touch

`touch [OPTION]... FILE...`

Het commando *touch* commando wordt in eerste instantie gebruikt om de timestamp van een bestand (laatst gewijzigde datum) te wijzigen. In tweede instantie zal het een leeg bestand aanmaken indien het bestand nog niet bestaat. De optie **-c** zal geen leeg bestand aanmaken indien het bestand nog niet bestaat. De optie **-r FILE** zal de timestamp van FILE gebruiken om de timestamp van de file die als parameter werd opgegeven te wijzigen. De optie **-t** snap ik niet. Het commando *touch -r /etc/passwd touchtest* zal een bestand touchtest aanmaken die dezelfde timestamp heeft als /etc/passwd.

1.11.8 comm

`comm [OPTION]... FILE1 FILE2`

Het commando *comm* neemt twee gesorteerde bestanden als argumenten en vergelijkt deze. wtf zijn die opties

1.11.9 diff

`diff [OPTION]... FILES`

Het *diff* commando zal bestanden lijn per lijn vergelijken. Opties zijn:

- **-i** | Niet hoofdlettergevoelig
- **-w** | Witruimten negeren in niet-lege lijnen
- **-B** | Lege lijnen negeren
- **-c#** | # aantal lijnen context rond verschillen
- **-u#** | Hetzelfde als -c#
- **-y** | De output in 2 kolommen weergeven

1.11.10 Message Digest

* [OPTION]... [FILE]...

Message Digest functies zijn one-way hashing functies. Commando's die hier onder vallen zijn: *md5sum*, *sha1sum*, *sha224sum* en *sha512sum*. Vul hhet * symbool op met één van deze vier commando's. Het commando *sha224sum input-file -c* TODOTODOTODO

1.11.11 paste

paste [OPTION]... [FILE]...

Paste zal verschillende bestanden lijn per lijn uitprinten. De **-d** optie stelt een delimiter in die in de output wordt verwerkt. De **-s** optie zegt dat de output sequentieel moet gebeuren. Dus eerst bestand 1, dan bestand 2 ...

1.11.12 join

join [OPTION]... FILE1 FILE2

Dit commando zal de lijnen van twee bestanden aan elkaar plakken als beide lijnen aan een predicaat voldoen. De opties zijn:

- **-t** | stelt de delimiter in
- **-1** en **-2** | Stelt het ON predicaat in
- **-o FORMAT** | Bepaalt het formaat van de uitvoer
- **-a 1 of 2** | Zal ook lijnen die niet gepaard kunnen worden uitprinten. 1 is enkel van bestand 1, 2 is enkel van bestand 2
- **-v** | zoals -a, maar beperkt de uitvoer tot lijnen zonder overeenkomst

1.11.13 split

split [OPTION]... [FILE [PREFIX]]

Het commando *split* zal een bestand opsplitsen in meerdere bestanden. De prefix zal voor elk van deze bestanden toegevoegd worden. Standaard is de prefix 'x'. De opties zijn:

- **-l#** | # aantal lijnen per bestand
- **-b#** | # aantal bytes per bestand
- **-c#** | Hetzelfde als -b maar zal niet stoppen in het midden van een lijn als het maximum is overschreden, maar zal deze lijn nog meenemen in het bestand.
- **-a#** | Bepaalt de lengte van de suffix. Standaard staat die op 2
- **-d** | Zal numerieke suffixes gebruiken

Zo zal het commando *split -d -l5 -a4 /etc/passwd rpx* om de 5 lijnen van */etc/passwd* een nieuw bestand aanmaken dat begint met *rpx* en daarna een numerieke suffix van 4 letters lang. De bestanden die kunnen aangemaakt worden zijn dan: *rpx0000*, *rpx0001*, *rpx0002*, *rpx0003*, ... , *rpxnnnn*.

1.11.14 echo

`echo [OPTION]... [STRING]...`

Het commando *echo* zal een lijn uitprinten. Standaard zal *echo* ook een newline(n) uitprinten. Dit kan je afzetten door de optie **-n** te gebruiken. *echo* herkent standaard geen escape characters. Om dit aan te zetten gebruik je de optie **-e**.

1.11.15 printf

`printf FORMAT [ARGUMENT]`

Het commando *printf* is een uitgebreide manier om tekst te printen naar de terminal. Het doel van *printf* is om een aantal argumenten mee te geven aan een formaatstring. Een formaatstring bestaat uit een aantal formaat karakters, net zoals in *C*. Zo zal *printf "%s tekst %d" "invoer" 5* de output "invoer tekst 5" hebben.

1.11.16 locate

`locate [OPTION]... PATTERN...`

Het commando *locate* zal bestanden zoeken die aan een bepaald patroon voldoen. Zo zal het commando *locate etc* alle directories en bestanden oplijsten waar de string *etc* in voorkomt. Als je wilt dat er geen directories getoond worden gebruik je de **-b** optie. Als je hoofdlettergevoeligheid wil uitschakelen gebruik je de **-i** optie. Standaard is *PATTERN* gewoon een eenvoudig patroon. Als je reguliere expressies wil gebruiken zoals je die in Perl gebruikt, moet je de **-r** optie meegeven.

1.11.17 find

Het commando *find* doet juist hetzelfde als *locate*, maar biedt meer mogelijkheden. Het enige voordeel van *locate* is dan ook dat het sneller is dan *find*. *find* is ook één van de zeldzame programma's dat de meeste opties in long-option notatie heeft. De belangrijkste opties zijn:

- **-maxdepth levels** | Zal only *levels* diep gaan in het bestandssysteem
- **-mount** | Ga niet in andere bestandssystemen
- **-anewer filename** | Enkel bestanden die later geopend zijn dan *filename*
- **-cnewer filename** | Enkel bestanden waarvan de permissies later gewijzigd zijn dan *filename*
- **-newer filename** | Enkel bestanden die later gewijzigd zijn dan *filename*

- `-nmin n` | Bestanden die n minuten geleden gewijzigd zijn
- `-mtime n` | Bestanden die n dagen geleden gewijzigd zijn
- `-uid u` | Bestanden waarvan u , een user ID, de eigenaar is
- `-user u` | Bestanden waarvan u , een username, de eigenaar is
- `-gid g` | Bestanden waarvan g , een group ID, de eigenaar is
- `-group g` | Bestanden waarvan g , een group name, de eigenaar is
- `-nouser` | Bestanden die geen gebruiker hebben
- `-nogroup` | Bestanden die geen groep hebben
- `-name n` | Bestanden waarvan n deel uitmaakt van de bestandsnaam
- `-iname n` | Bestanden waarvan n deel uitmaakt van de bestandsnaam, niet hoofdlettergevoelig
- `-perm mode` | Bestanden die aan een bepaald permissiepatroon voldoen
- `-size (\pm)n(b, k, M, G)` | Bestanden die exact n groot zijn, bestanden die groter zijn dan $+n$ of bestanden die kleiner zijn dan $-n$
- `-type t` | Bestanden die van type t zijn. t kan **d**(irectory), **l**(ink) of **f**(ile) zijn. Zie manpagina voor meer

Deze opties kunnen vanzelfsprekend gecombineerd worden om heel specifiek te gaan filteren. Hier volgt een voorbeeld

```
find /usr/share
-name '*todo*'
-size +10k
-mtime +200
-printf '\%8s \%p'
```

Dit commando zal alle bestanden vinden waarin *todo* vermeld staat in de map `/usr/share`. Verder worden enkel bestanden ouder dan 200 dagen en groter dan 10 kilobyte in achtning genomen. De `printf` zal eerst het aantal bytes van het bestand en dan de naam van het bestand uitprinten. Merk op dat `printf` hier niet dezelfde conventies volgt als die van *C*. Gebruik de manpagina om meerdere formaatkarakters te bekijken.

1.12 Intermediate Commands

1.12.1 cat

Het commando *cat* zal bestanden één voor één uitprinten. Opties zijn:

- `-n` | De lijnen worden ook genummerd

- -b | Zelfde als -n behalve dat blanco lijnen niet genummerd worden
- -s | Opeenvolgende blanco lijnen als één lijn beschouwen
- -v | Alle controletekens worden getoond als ^??? waarbij ??? een specifiek teken is
- -T | Tabs zullen als ^I getoond worden
- -E | Zal een \$ symbool plaatsen achter elke lijn

1.12.2 tac

Het commando *tac* is het omgekeerde van *cat*. Het zal de bestanden dus in omgekeerde volgorde uitprinten. De opties zijn hier wel beperkt. De optie **-s** **STRING** laat een lijnscheidsteken **STRING** toe. Dit vervangt dan de newline lijnscheidsteken. De optie **-r** Laat toe dat deze lijnscheidsteken een reguliere expressie is. **-r** kan dus enkel gebruikt worden als **-s** ook aanwezig is.

1.12.3 rev

Het commando *rev* is gelijkaardig aan *cat*, maar zal elke lijn omgekeerd uitprinten. De lijn *Dit is een stukje tekst* wordt dan *tsket ejkuts nee si tiD*. Dit commando heeft geen nuttige opties.

1.12.4 wc

Het commando *wc* zal lijnen, woorden en bytes tellen en uitprinten. Zonder opties print *wc* deze drie gegevens uit. Wil je enkel bytes, woorden of lijnen gebruik je respectievelijk de **-c**, **-w** of **-l** optie. Deze opties kunnen gecombineerd worden om bv. enkel het aantal woorden en lijnen te hebben. De optie **-L** toont ook de lengte van de langste lijn. Je kan deze optie in combinatie met het commando *strings* (dat later aan bod komt) gebruiken om de effectieve lijn uit te printen: *strings -n \$(wc -L /etc/passwd)*.

1.12.5 head

Het commando *head* zal de eerste 10 lijnen van een bestand uitprinten. De optie **-n#** zal # aantal lijnen uitprinten in plaats van de default 10. De optie **-c#** zal # aantal bytes uitprinten. Als je # laat voorafgaan door een minteken (-), dan zal *head* alle lijnen behalve de laatste # uitprinten.

1.12.6 tail

Het commando *tail* heeft de omgekeerde werking van *head* en zal dus enkel de laatste 10 lijnen van een bestand uitprinten. *tail* kent dezelfde **-n#** en **-c#** optie. Als de # wordt voorafgegaan door een minteken (-), dan zal *tail* vanaf lijn/byte # beginnen printen. *tail* kent ook de optie **-f**. Dit zal een bepaald

bestand in de gaten houden en zal naar de terminal printen wanneer dit bestand gegroeit is in bytes. De optie **-F** is een hardere versie in de zin dat het geen foutmeldingen zal opleveren en dat het zal blijven proberen het bestand te openen indien het nog niet gelukt is.

1.12.7 nl

Het commando *nl* zal het aantal lijnen in een bestand tellen. De optie **-v#** zal de lijnnummering starten vanaf **#**. De optie **-ba** zal blanco lijnen ook nummeren. De optie is hier in feite **-b** en het argument van deze optie is **a**. In de man pagina staat er ook nog **t**, wat enkel niet lege lijnen zal nummeren en **n**, wat geen lijnen zal nummeren. **t** is de default waarde en **n** heeft geen nut als je dit programma gebruikt want je gebruikt dit namelijk om lijnen te nummeren. De optie **-n** werkt op dezelfde manier. Meestal zal je **-nrz** gebruiken. Hier worden leading zeros aan de lijnummers toegevoegd en de tekst wordt rechts gealigneerd.

1.12.8 cut

Het commando *cut* zal specifieke secties van elke lijn van een bestand verwijderen en deze uitprinten naar de terminal. De eerste optie is **-c**. Dit zal een range van kolomnummers(aantal karakters) teruggeven. Het commando *cut -c5-10 /etc/passwd* zal enkel karakter op positie 5 tot en met 10 uitprinten voor elke lijn van het bestand */etc/passwd*. Als je andere opties wilt gebruiken mag je de **-c** optie niet gebruiken. De andere optie is **-f**. Deze zal de veldnummers selecteren. Deze optie wordt best gebruikt met de **-d** optie, wat de delimiter instelt en de **-s** optie, dat enkel lijnen zal evalueren waarin de delimiter in voorkomt. Zo zal *cut -d ':' -f1,3 -s /etc/passwd* Het eerste en het derde veld van elke lijn in */etc/passwd* uitprinten naar de terminal. Merk dus op dat je ofwel de **-c** ofwel de **-f** optie met extensieopties gebruikt. Je kan ook nog **-complement** gebruiken, wat de selectie complementeert.

1.12.9 strings

Strings zal voor elk bestand dat meegegeven wordt, enkel de printbare characters uitprinten die minstens 4 karakters lang zijn. De eerste optie is **-n#**. De output zal enkel woorden bevatten die **#** karakters lang zijn. De optie **-t(d—x—o)** neemt een van de parameters die tussen haakjes staat mee. Deze optie zal voor elke lijn cumulatief het aantal karakters in het begin van de lijn printen. **d**, **x** en **o** stellen het aantal karakters respectievelijk voor als decimaal, hexadecimaal en octaal.

1.12.10 sort

Het commando *sort* zal een bestand sorteren. De opties zijn:

1. **-t** | Wijzigt de delimiter in.

2. -k | Stelt de sorteersleutel in. Deze wordt gevolgd door het veld dat je wil sorteren (afhankelijk van de delimiter)
3. -f | Niet hoofdlettergevoelig
4. -n | Numeriek sorteren in plaats van alfabetisch
5. -r | Sorteervolgorde omkeren
6. -o FILE | Zal de output wegschrijven naar FILE. Dit mag hetzelfde zijn als het invoerbestand
7. -c | Controleert eerst ofdat de invoer al gesorteerd is en geeft een foutmelding als dit wel zo is

Met behulp van deze commando's kan je `/etc/passwd` bijvoorbeeld op user ID sorteren: `sort -n -t -k3,3 /etc/passwd`. Je kan dit commando dan doorpijpen naar `cut` om enkel de naam en de ID uit te printen: `sort -n -t -k3,3 /etc/passwd | cut -d ' ' -f1,3`.

Hoofdstuk 2

Devices

2.1 Devices Files

Devices worden in Unix op twee manieren gerepresenteerd. Enerzijds in de `/dev` folder als een *device file*. Dit wordt voornamelijk gebruikt om programmatisch deze apparatuur te bevragen. Anderzijds in het `/sys` virtueel bestandssysteem. Dit wordt dan gebruikt om detailinformatie per device te verzamelen.

Bij het uitvoeren van `ls -ladH` in de `/dev` folder krijg je output gelijkaardig aan `ls -l`. De `H` optie zorgt ervoor dat er bij elk bestand extra informatie staat met betrekking tot het device file type (**b**lock, **c**haracter, **s**ocket of **p**ipe). Er is ook een major en een minor getal. Het major getal zegt welke driver er gebruikt wordt voor deze file. Het minor getal is een instantie per driver om onderscheid te kunnen maken tussen de verschillende devices. Device files kunnen bewerkt worden zoals eender welke file (bv `cat` en `echo`).

2.1.1 Block devices

Typisch aan een block device is dat ze dienen voor massa opslag. Daarom kan het ook verschillende buffergroottes aannemen. Voorbeelden van block devices zijn harde schijven en read-only drivers voor CD's. Een write driver is geen block device maar een character device. Loop devices zijn "valse" devices die zich gedragen als een block device. Een voorbeeld van een loop device is een `.iso` bestand.

2.1.2 Character devices

Character devices hebben in tegenstelling tot block devices een vaste buffergrootte. Dit is de grootste restrictie van een character device. Voorbeelden van character devices zijn terminals en pseudo-devices. Pseudo-devices worden in de volgende paragrafen toegelicht. Terminals worden besproken in sectie ??

/dev/null

Er zijn een aantal nuttige pseudo-devices die kunnen gebruikt worden tijdens het maken van shell-scripts. De eerste is **/dev/null**. /dev/null kan worden gebruikt om bepaalde uitvoer te vernietigen zoals in het volgende voorbeeld:

```
ls -l {01..09} 2>/dev/null | wc -l
```

In dit geval bestaan bestanden 08 en 09 niet, wat een foutmelding zal geven, maar aangezien het standaard foutenkanaal(2) wordt omgeleid naar /dev/null zal deze niet op de terminal komen maar direct verwijderd worden. Als we nu geïnteresseerd zijn om enkel de fouten te tonen kan het volgende commando gebruikt worden:

```
ls -l {01..09} 2>1 1>/dev/null | wc -l
```

Hier wordt het foutkanaal omgeleid naar het standaard invoerkanaal. Het standaard invoerkanaal wordt omgeleid naar /dev/null. Op het eerste zicht lijkt dit raar, maar de argumenten worden in deze instantie in omgekeerde volgorde uitgevoerd.

Beschouw het volgende commando:

```
grep a $(ls 0?)
```

Op normale wijze zal dit commando elk bestand overlopen die begint met een 0 en daarna nog een karakter heeft en deze bestanden zoeken op de letter 'a'. In het geval dat er geen bestanden gevonden zijn zal grep blijven hangen en verwacht hij input. Dan moet je zelf invoer geven wat zeker niet gewenst is. Dit kan opgelost worden door /dev/null als parameter mee te geven.

```
grep a $(ls 0? 2/dev/null) /dev/null
```

Het laatste voorbeeld zal gebruik maken van het *xargs* commando. De bedoeling is om een aantal bestanden te overlopen en dan als uitvoer geven welk bestand er ergens een letter 'a' heeft.

```
grep ls -l 0? | xargs -n3 grep a
```

Dit geeft het volgende als uitvoer:

```
01 plya
fzaj
```

In bestand 01 zit er dus een letter a, maar we weten niet in welk bestand de andere letter a zit. Als we de -t optie toevoegen aan het *xargs* commando krijgen we dit als output:

```
grep 01 02 03
01 plya
grep 04 05 06
grep 07
fzaj
```

Aangezien we groepjes van drie hebben en maar zeven bestanden hebben zal de laatste groepering maar één element hebben. Het *grep* commando zet er dan niet meer bij van welk bestand dit komt. Dit kan ook weer opgelost worden met */dev/null* op de volgende manier:

```
ls -l 0? | xargs -n3 -t grep a /dev/null
```

Dit geeft als uitvoer:

```
01 plya
07 fza j
```

/dev/zero

/dev/zero is een device dat null characters (0x00 of whitespace) zal genereren. Bekijk het volgende voorbeeld:

```
head -c 60 < /dev/zero | tr '\0' '-' > file
```

Het *head* commando zal 60 whitespace characters uit */dev/zero* halen. Het *tr* commando zal deze whitespaces omvormen tot een streepje en uiteindelijk zal deze uitvoer in het bestand *file* worden geplaatst. Als je dit bestand opent zal je zien dat de inhoud 60 streepjes bevat.

/dev/random* en */dev/urandom

/dev/random en */dev/urandom* zijn beide randomgenerators. Het verschil ligt erin dat */dev/random* een beperkte pool heeft. Deze pool is snel leeg en duurt minuten om terug te hervullen, wat de randomgenerator tijdelijk onbeschikbaar maakt. */dev/urandom* heeft dit probleem niet, maar de randomgenerator van */dev/urandom* is niet zo sterk als die van */dev/random*. Het volgende voorbeeld zal random IPv4 adressen genereren:

```
head -c 80 < /dev/urandom | od -An -w4 -t u1
                        | tr -s ' ' ':'
                        | cut -c 2-
```

Gelijkaardig voor random IPv6 adressen:

```
head -c 160 /dev/urandom | od -An -w16 -t x2
                        | tr ' ' ':'
                        | cut -c 2-
```

File descriptors

Een filedescriptor is een getal dat verwijst naar een bestand om zo read en write operaties mogelijk te maken. Om een lijst van alle filedescriptors op te vragen wordt het commando *ls -lad /proc/PID/fd/** gebruikt. het PID is het procesid van de huidige terminal. Dit kan opgevraagd worden met *echo \$\$*. De output zal standaard vier filedescriptors tonen. 0 → standaard input, 1 → standard

output, 2 → standard error en 255 → filedescriptor voor de andere drie te herstellen (specifiek voor bash). Een file descriptor aanmaken kan in drie modi gebeuren:

```
exec 3 < file
exec 4 > file
exec 5 <> file
```

Er wordt gebruik gemaakt van het *exec* commando. Dit commando zal een opdracht voor de levensduur van het venster in achtting houden. Op die manier kunnen we aan de filedescriptors op eender welk moment, zolang we in dezelfde terminal zijn. < zal een filedescriptor aanmaken in leesmodus, > in schrijfmodus en <> in beide modi. Het getal voor deze operatoren is het nummer van de filedescriptor.

Bij het lezen van een bestand zonder filedescriptors zal er geen pointer bijgehouden worden van de huidige lijn. Dit heeft als gevolg dat de volgende code telkens de eerste lijn van een bestand zal printen:

```
while read lijn < file; do echo $lijn; done
output:
een
een
een
```

Filedescriptors hebben dus een pointer naar de huidige locatie in een bestand. Als het vorige commando wordt uitgevoerd met een filedescriptor, zullen alle lijnen overlopen worden:

```
while read lijn < &3; do echo $lijn; done
output
een
twee
drie
```

Een file descriptor kan ook schrijven naar een bestand. *echo tekst > &4* zal het stukje 'tekst' toevoegen aan filedescriptor 4. De originele tekst zal niet vervangen worden zoals bij een echo commando met een normaal bestand. Voor schrijfoperaties zijn filedescriptors ook een pak performanter dan echo commando's

2.2 Terminals

```
//progressindicator
((x=2**30));tput sc; while ((x /= 2)); do tput rc; echo -n $x; sleep 0.5; done
```

Hoofdstuk 3

Shell Scripts

3.1 Shell Script Basics

Vooraleer er een shell commando wordt uitgevoerd worden er een aantal stappen overlopen. Dit onderdeel zal de belangrijkste stappen uit dit proces toelichten.

3.1.1 2. Brace Expansion

Brace Expansion is een mechanisme dat random strings kan laten genereren. Beschouw het volgende voorbeeld:

```
echo {0..9}{a..f}
```

OUTPUT

```
0a 0b 0c 0d 0e 0f 1a 1b 1c 1d 1e 1f
2a 2b 2c 2d 2e 2f 3a 3b 3c 3d 3e 3f
4a 4b 4c 4d 4e 4f 5a 5b 5c 5d 5e 5f
6a 6b 6c 6d 6e 6f 7a 7b 7c 7d 7e 7f
8a 8b 8c 8d 8e 8f 9a 9b 9c 9d 9e 9f
```

Dit geeft dus een lijst van alle mogelijke combinaties waaruit het eerste karakter bestaat uit een cijfer en het tweede karakter een letter van a tot d. Het volgende voorbeeld zal dit vermijden:

```
echo {{0..9},{a..f}}
```

OUTPUT

```
0 1 2 3 4 5 6 7 8 9 a b c d e f
```

Er is één probleem met brace expansion en dat is dat het vroeg komt in het proces. Dit zorgt ervoor dat het volgende voorbeeld niet zal werken:

```
declare x = 5
declare y = 20
echo {$x..$y}
```

OUTPUT

```
{$x..$y}
```

Om dit op te lossen wordt het *eval* commando gebruikt

```
declare x = 5
declare y = 20
eval "echo {$x..$y}"
OUTPUT
5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

3.1.2 5. Command Substitution

Command substitutie laat toe om een commando uit te voeren en de uitvoer van dit programma als argumenten van een ander programma te laten gelden. Een eenvoudig voorbeeld is `vi $(ls -la)`. Dit zal de uitvoer van `ls -la` openen in de `vi` editor.

3.1.3 6. Process Substitution

Process substitution is gelijkaardig aan command substitution, alleen zal de uitvoer in een tijdelijk bestand terecht komen en dit bestand wordt dan gebruikt als argument bij een ander commando. Het commando `diff` verwacht twee bestanden om te vergelijken. Als je de bestanden nog niet hebt is het gemakkelijker om aan process substitution te doen: `diff <(sort bestand1) <(sort bestand2)`. Process substitution wordt dus voorafgegaan met een `|` terwijl commando substitution voorafgegaan wordt met een `$`.

3.1.4 ? Arithmetic Substitution

Arithmetic substitution laat toe om een argument als numerieke waarde te beschouwen in plaats van een stukje tekst.

```
echo $(( 5 + 3 ))      \  8
echo 5 + 3             \  5 + 3
```

```
for ((i=1; i<=$#; i++));
do echo $i;
```

```
ARRAY :    ${@:i:1}
POINTER:  $$!1
```

3.1.5 ?. Parameter Expansion

Parameter Expansion behandelt de procedure om een waarde van een variabele te krijgen. Dit kan toegepast worden op vele voorbeelden. Stel dat de variabele `x=abcdefg`. Als we `echo ${x}0000` uitvoeren krijgen we `abcdefg0000` als uitvoer. `echo ${#x}` zal de lengte van de variabele tonen, in dit geval is dit `7`. Hier kunnen ook reguliere expressie gebruikt worden. Om een bepaald stukje weg te laten

kan bijvoorbeeld `echo ${x#*e}` gebruikt worden. Dit heeft als resultaat `fg`. Om het omgekeerde effect te bekomen gebruik je `echo ${x%e*}` wat als uitvoer `abcd` heeft.

In het volgende voorbeeld gebruiken we de variable `y=abcd0000ef00gh`. Stel dat we eerste rij van nullen en alles daarvoor willen verwijderen. Dan gebruiken we `echo ${y#*0}`. Dit geeft als uitvoer `ef00gh`. Om dit voor alle nullen (globaal) te doen gebruik je `echo ${y###*0}`. Dit geeft als uitvoer `gh`.

`${PWD%*}` → ouderdirectory

`${PWD##${PWD%*}}` → huidige directory

3.2 Functies

Functies laten toe om code te hergebruiken. Een functie wordt in Unix gedeclareerd op de volgende wijze: `f()`. `f` is hier de functienaam die gelijk wat kan zijn. In de open haakjes mag er niets staan, dit is een indicatie dat het om een functie gaat. De volgende functie zal het aantal lijnen tellen van een bepaald bestand:

```
f() { wc -l file ; }
```

Het is ook mogelijk om parameters mee te geven aan een functie. De shell zal deze parameters dan opvullen in `$1`, `$2`, ... `$n`. Het volgende voorbeeld zal de twee meegegeven parameters uitprinten:

```
g() { echo $2 $1 ; }
```

Deze functie kan nu aangeroepen worden via `g één twee drie`. Dit zal als output `twee één` hebben. Overbodige parameters zijn dus geen probleem, ze worden alleen niet gebruikt. Tot slot kunnen functies ook een returnwaarde geven, alhoewel dit meestal niet nuttig is. Bekijk het volgende voorbeeld:

```
h() { return 200 ; }
```

Om deze returnwaarde nu te bekijken gebruik je `$?`, waarin 200 zal staan. Het nadeel aan deze returnwaarde is dat het maximum 256 kan bevatten en dus ook geen negatieve getallen.

Normaal gezien zijn variabelen en parameters passed by value. Dit wil zeggen dat de variabele zelf niet aangepast zal worden maar enkel de waarde. Als demonstratie wordt het volgende voorbeeld gebruikt.

```
i() { echo tijdens: $x ; }
```

Als we de variabele `x` instellen op `ervoor` en we voeren dan de volgende commando ketting uit: `echo ervoor: $x ; i ; echo erna: $x`. Dit zal als output `ervoor: ervoor tijdens: ervoor erna: ervoor` hebben. Als we de functie nu aanpassen naar `i() { x=tijdens; echo tijdens: $x }`, zal de output `ervoor: ervoor tijdens: tijdens erna: tijdens` zijn. Om de variabele alleen lokaal te gebruiken wordt het keyword `local` of `declare` gebruikt. Dus `i() { local x=tijdens; echo tijdens: $x }`. Dit geeft als output `ervoor: ervoor tijdens: tijdens erna: ervoor`.

Om een lijst van alle functies op een toestel op te vragen wordt het commando `declare -F` gebruikt. Om de source te bekijken van bv. het commando `_tilde` wordt `declare -fp _tilde` gebruikt.

3.3 Variabelen

Het instellen en gebruik van variabelen is bij Bash heel beperkt. Om een variabele in te stellen kan ofwel `x=abc` of `declare x=abc` gebruikt worden. Het voordeel aan `declare` is dat er nog extra opties kunnen meegegeven worden, maar zoals eerder vermeld is dit heel beperkt. Een paar voorbeelden zijn `-l` en `-u`, die respectievelijk de inhoud van de variabele naar lowercase of uppercase brengt. Om een variabele te verwijderen wordt `unset x` gebruikt.

In het geval dat we een variabele `z` instellen als `343` en we wensen hierbij een ander getal bij op te tellen zou het logisch lijken om `z+=57` te doen. Echter deze operatoren zullen concateneren wat de inhoud verandert in `34357`. Om effectief getallen op te tellen moet `((z+=57))` gebruikt worden, wat 400 oplevert.

3.3.1 read

Er wordt gebruik gemaakt van het `read` commando om demonstraties met variabelen te tonen. `read` zal een lijn uitlezen en deze in de `REPLY` variabele steken. Dus als we `read j competitors.csv` uitvoeren en we kijken in de `REPLY` met behulp van `declare -p REPLY` zien we de eerste lijn van dit bestand staan. Het `read` commando kan ook intern splitsen op een scheidingsteken. Als we `read a b c j competitors.csv` uitvoeren zal de eerste lijn gesplit worden. Om te weten hoe er gesplit werd moet je individueel de variabelen `a`, `b` en `c` bekijken. Wat opvalt is dat het scheidingsteken een spatie is. Dit komt door de `IFS` variabele. Als we `hexdump -C jii "$IFS"` uitvoeren krijgen we de variabele die de standaard lijnscheidingstekens bepaalt. Default staat die op een spatie (20), een tab (09) en een newline (0a).

De `IFS` variabele kan voor één lijn verandert worden door bv:

```
IFS=\; read a b c < competitors.csv
```

Dit zal de `IFS` variabele opvullen met enkel een komma-punt teken. Na het uitvoeren van de bovenstaande regel wordt die terug op de default waarde ingesteld. Als je nu voor meerdere lijnen een andere lijnscheidingsteken wil gebruiken moet je een functie gebruiken:

```
f() {
    { read a b c < competitors.csv } ;
    { read a b c < competitors.csv } ;
}
IFS=\; f
```

Als we geen invoer geven aan `read`, dus op deze manier: `read y`. Zal `read` in interactieve modus starten. Hij beschouwt dit dan als invoer en zal het schrijven naar de variabele `y`.

De optie `-nx`, waarbij x een natuurlijk getal is, zal de invoer beperkt worden tot x karakters. Hierbij stopt een lijnscheidingsteken ook de invoer. Als de optie `-Nx` wordt uitgevoerd, zal een lijnscheidingsteken ook meetellen als een karakter. Ook kan er een timeout gespecificeerd worden met de `-tx` optie.

3.4 Foutcodes

3.5 Arrays

3.5.1 Numeriek

Een array kan in bash op twee manieren gedeclareerd worden. Numeriek of associatief. Een numerieke array wordt enkel geïndexeerd met getallen. Er kan direct gebruik gemaakt worden van een numerieke array zonder dat hij eerst gedeclareerd moet worden. `t[5]=abc` zal dus een array t aanmaken dat op index 5 de string *abc* bevat. Merk op dat alles aan elkaar plakt in deze instructie. Om nu de informatie die in de array zit te bekijken kan je `declare -p t`. Een element toevoegen kan door gewoon een andere index te gebruiken: `t[8]=def`. Je kan ook meerdere elementen tegelijk toevoegen: `t+=([2]=ghi [7]=jkl)`. Je kan ook de index weglaten: `t+=(mno pqr)`. Hier zal de volgende hoogste index gebruikt worden, dus index 9 en 10. Tot slot kan ook nog de uitvoer van een programma gebruikt worden: `t+=($(cat file))`. Hier wordt de default delimiter gebruikt.

Met `unset "t[2]"` verwijder je het tweede element van de array. Om de inhoud van een index te bekijken gebruik je `echo ${t[5]}`. Om alle elementen te tonen gebruik je `"${t[@]}"` en voor alle sleutels gebruik je `"${!t[@]}"`. Om te checken of dat een bepaalde index een waarde heeft gebruik je het `-v` programma op volgende manier: `[[-v t[8]]] && echo bestaat || echo bestaat niet`.

3.5.2 Associatief

Associatieve arrays moeten op voorhand gedeclareerd worden door middel van `declare -A t`.

3.6 Conditionele logica

3.6.1 If tests

Bash kent een andere manier om te testen aan een specifieke voorwaarde. De `||` en `&&` tekens stellen respectievelijk OR en AND voor. De voorkeur om te testen gebeurt op volgende manier:

```
cmd && { cmd1 ; cmd2 ; ... ; cmd9 ; } || { cmda ; cmdb ; ... ; cmdz ; }
```

Dit zal de eerste reeks commando's uitvoeren als *cmd* een true oplevert en de tweede reeks commando's als het een false oplevert.

3.6.2 Case

SYNTAX: `case WORD in [PATTERN [| PATTERN]...] COMMANDS ;;]... esac`

VOORBEELD:

```
case 458 in
    *1*) echo 1 ;;
    *1*) echo 2 ;;
    *1*) echo 3 ;;
    *1*) echo 4 ;;
    *1*) echo 5 ;;
    *1*) echo 6 ;;
    *1*) echo 7 ;;
    *1*) echo 8 ;;
    *1*) echo 9 ;;
esac
```

Een bepaalde case kan gestopt worden met 3 symbolen

1. `;; -i` Zal stoppen nadat er één case voldaan werd
2. `;;& -i` Zal alle cases die voldoen uitvoeren
3. `;& -i` Voert alle cases uit na de eerste case die voldaan werd

Het is mogelijk om deze symbolen te combineren, maar wordt sterk afgeraden.

3.7 For loops

SYNTAX: `for NAME [in WORDS ...] ; do COMMANDS; done` VOORBEELD:

```
for i in "a b c d" ; echo $i; done
```

3.8 While loops

SYNTAX: `while COMMANDS; do COMMANDS; done` VOORBEELD:

```
while read lijn ; do $lijn >> log ; done bestand || NOG EENS BEKIJKEN KLOPT N
```

Verbeterde versie met filedescriptors

```
exec {fd} < bestand
while read lijn <&${fd} ; do echo $lijn ; done
exec {fd} <&-
```

3.8.1 Teken per teken verwerken

```
while read -n1 char ; do ... ; done <<< "$string"
```

3.8.2 Verwerken uitvoer opdracht

```
while read lijn ; do ... ; done < <(opdracht)
```

3.9 Tijdelijke bestanden

Tijdelijke bestanden kunnen aangemaakt worden met het **mktemp** commando. Dit commando zal een bestand aanmaken met een naam dat aan een specifiek patroon voldoet en deze naam teruggeven. **mktemp 123XXX456XXX** zal enkel de laatste vier X'en vervangen door willekeurige letters en cijfers.

3.10 Here Documents en Strings

Er bestaan twee manieren om op het moment zelf invoer te creëren dat als een bestand beschouwd wordt.

```
x='een  
twee  
$(ls)  
drie '  
rev <<< "$x"
```

of

```
rev << EOF een  
twee  
$(ls)  
drie  
EOF
```

De voorkeur gaat naar de eerste methode.

3.11 Signalisering

Het commando **kill** wordt gebruikt om een signaal naar een proces te sturen. Om een lijst van alle signalen te verkrijgen gebruik je **kill -l**. Om signalisering duidelijk te maken schrijven we eerst een script dat priemgetallen zal zoeken.

```
i=0;  
while ((++i));  
do t=$(factor $i);  
(( ${#t[@]}=2 )) && echo $i;  
done
```

Als je dit script zal uitvoeren zal het een getal naar het scherm printen als dit een priemgetal is. We willen nu dat dit script het laatste priemgetal uitprint als we dit programma beëindigen met CTRL + C. Het programma wordt als volgt aangepast:

```
trap 'print $p, $n' 10;
i=0; n=0;
while ((++i));
do t=$(factor $i);
(( ${#t[@]}=2 )) && echo $i;
{p=$i; ((n++))
done;
```

We kunnen nu dit programma stoppen met CTRL + C waarbij het programma het laatste priemgetal uitschrijft. Dit programma kan ook gestopt worden via een andere terminal. Om het pid te achterhalen gebruik je **ps tree -p | grep bash**. Daarna gebruik je dit pid op volgende manier: **kill -10 pid**.

Om te debuggen wordt het DEBUG signaal gebruikt op volgende manier:

```
trap '$i' DEBUG
i=0;n=0;
while ((++i));
do f=$(factor $i);
(( ${#f[@]}==2 )) && { set -x; ((n++)); p=$i; set +x;} done;
```

In het geval dat set -x en set +x voorkomen zal het DEBUG signaal in werking treden. Verder zal er ook aales wat tussen set -x en set +x staat uitgeprint worden. Het volstaat dus ook om gewoon set -x; set +x; te hebben, al is het maar voor het DEBUG signaal te laten werken