



Linux voor Bio-Informatici

Deze reader is gebaseerd op de HAN Linux Reader van de Informatica en Communicatie Academie van de HAN. Verbeteringen en aanpassingen aan Bio-Informatica, Martijn van der Bruggen, Esther Kok.

Inhoud

1. Inleiding Linux voor Bio-Informatica	3
Algemene inleiding	3
Het besturingssysteem	3
UNIX	3
Linux	3
Linux en Bio-Informatica	4
2. Starten met Linux: commando's	5
Linux account	5
De eerste schreden	6
3. Bestanden manipuleren	8
4. Globben, tabben en greppen	11
Tabben	11
Globbing *?[^]	11
Reguliere Expressies	12
Grep	12
5. De Manual Pages	13
De UNIX manual: 'man'	13
De shell	15
Info, de opvolger van man	17
6. Tekst bewerken: VI	18
7. Omleidingen en pijpleidingen	22
Invoer en uitvoer omleiden	22
Pipeline	24
8. rwxr-xr-- en andere permissies	26
Sommige dingen mogen niet	26
De permissies veranderen CHMOD	28
Permissie om te verwijderen	29
9. Shell scripts	31
Commando's combineren	31
Shell variabelen	32
Variabelen in een script	33
10. File Transfer Protocol	35
11. En nu verder	39

1. Inleiding Linux voor Bio-Informatica

Algemene inleiding

Deze stoomcursus probeert je in korte tijd een gevoel voor de kracht van UNIX systemen zoals Linux te geven, en een beetje overzicht van haar eigenschappen. We willen je uitnodigen te experimenteren, dingen uit te zoeken en na te denken over oplossingen en eigenschappen. Om het beste te maken van de stoomcursus raden we je aan om de dingen zoveel mogelijk zelf (alleen dus) te doen, maar de denk- en puzzelingen samen met anderen.

Het besturingssysteem

Een besturingssysteem (in het Engels operating system of afgekort OS) is het programma (meestal een geheel van samenwerkende programma's) dat bij opstarten van de computer als eerste in het geheugen geladen wordt, en dat de functionaliteiten aanbiedt om andere programma's te laten uitvoeren. Er zijn erg veel besturingssystemen die we kunnen gebruiken op computers. Het bekendste en meest gebruikte besturingssysteem voor desktop en laptops is Windows.

UNIX

UNIX is een besturingssysteem vergelijkbaar met Windows maar dan juist bedoelt voor grote bedrijfsservers. Bij grote bedrijfsservers moet je denken aan bijvoorbeeld de machines van Amazon waar de gegevens van honderdduizenden klanten en producten worden bijgehouden en verwerkt. Er zijn talloze versies van UNIX ontwikkeld en meestal is dit heel prijzige software. Voor bedrijfsservers worden meestal UNIX-achtige besturingssystemen gebruikt.

Linux

Linux is een besturingssysteem dat afgeleid is van UNIX het lijkt daardoor ook erg sterk op UNIX. Linux is echter vrije software en open source. Vrije software betekent dat voor het gebruikt niets betaald hoeft te worden. Open source geeft aan dat de code (source) voor iedereen vrij toegankelijk is (open). Iedereen mag in de broncode van Linux kijken en er verbeteringen in aanbrengen. Daarna mag de code ook weer onder eigen naam gedistribueerd worden mits aan een aantal voorwaarden is voldaan. Zo zijn er tal van Linux

distributies verschenen die allemaal hun eigen karakteristieken kennen. Een groot aantal organisaties maakt gebruik van Linux, zo ook Amazon. Dit aantal neemt alleen maar toe.

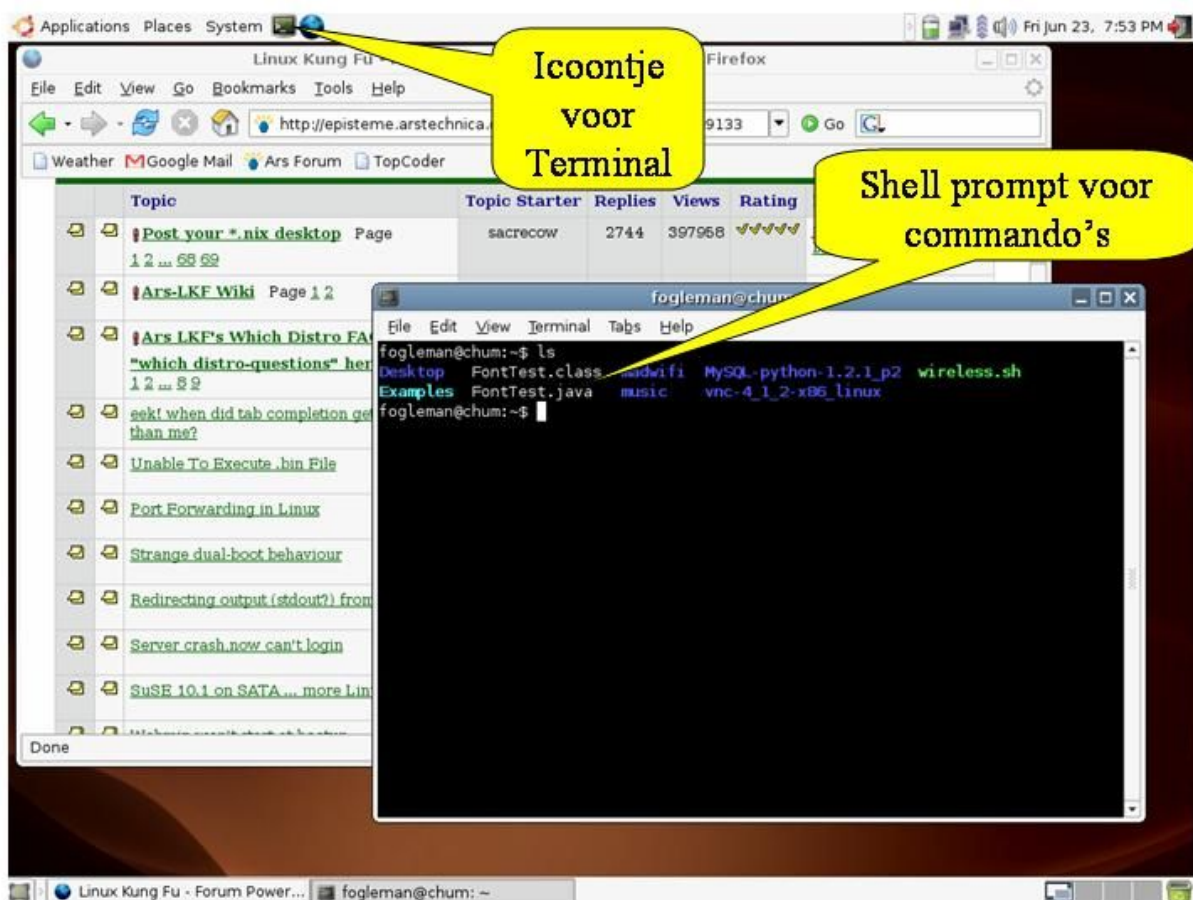
Linux en Bio-Informatica

Binnen de bio-informatica maken we veel gebruik van Linux. Juist vanwege het open karakter biedt het veel mogelijkheden om er bio-informatica applicaties op te draaien. Veel bio-informatica applicaties zijn ontwikkeld voor Linux of vaak gebaseerd op Java. Alle Java applicaties zijn platform onafhankelijk en kunnen dus op zowel Linux, UNIX als Windows besturingssystemen draaien.

2. Starten met Linux: commando's

Linux account

Om in te loggen op de Linux machines wordt gebruik gemaakt van aparte login namen en wachtwoorden. Deze krijg je van de docent. Log in op een Linux machine en start een Linux Shell (Terminal sessie). In deze Linux Shell is het mogelijk om Linux commando's op de prompt op te geven. Linux is vergelijkbaar met Windows. Linux kent echter een uitgebreide manier om vanaf de command prompt (Wikipedia) opdrachten op te geven.



Figuur 3: Screenshot eigen werk auteur

De eerste schreden

Om je plaats in het filesysteem te bepalen geef je het volgende commando:

```
pwd
```

De uitvoer is het volledige pad naar de huidige directory. Daar zit je dus. Omdat je net ingelogd bent is dit je 'home-directory'.

UNIX is een multi-user systeem, en elke gebruiker krijgt, naast een loginnaam en een wachtwoord, een eigen plaats om bestanden op te slaan, de home directory. De naam van die directory is meestal dezelfde als loginnaam van de eigenaar. Vaak kunnen we het pad naar je homedirectory afkorten met '~' (een tilde), en het pad naar de homedirectory van een ander met '~login' waarbij login dan de loginnaam van die ander is.

Het 'ls' commando toont de namen van de bestanden in huidige directory. Nu je dit doet terwijl het de eerste keer is dat je inlogt zal je nog niet veel zien. Maar probeer eens

```
ls -a
```

Dit zou al meer namen moeten opleveren, waarvan de meeste namen met een punt beginnen.

Onder UNIX is het de gewoonte om bestandsnamen die met een punt beginnen als 'onzichtbaar' te beschouwen. Zonder de '-a' optie zal 'ls' ze ook niet tonen. Veel programma's maken zulke onzichtbare bestanden in je homedirectory aan om instellingen en voorkeuren op te slaan. Eén ervan, '.bashrc', bevat commando's die gestart worden elke keer als je inlogt. Hier zullen we later terug komen.

Het 'cd' commando gebruik je om naar andere directories te gaan. Ga één directory omhoog in de hiërarchie met

```
cd ..          # in tegenstelling tot Windows/DOS moet er een spatie tussen.
```

Een 'ls' nu zal een lijst met andere home directories van andere gebruikers opleveren. Je kunt nu terug naar je homedirectory gaan met het cd commando, maar interessanter is het volgende:

```
cd /usr/bin
ls
```

Dit is een directory waar veel programma's worden bewaard. Neem even de tijd om met behulp van 'cd' en 'ls' rond te neuzen in de subdirectories van /usr/bin. Doe ook even

```
cd /  
ls
```

om de top van de directory hiërarchie te zien.

Onder UNIX is '/' het teken dat directorynamen in een pad scheidt, niet '\' zoals onder DOS. Paden die beginnen met '/', zoals '/usr/doc' worden '**absoluut**', of '**volledig**' genoemd, en beschrijven het hele pad van de top van de hiërarchie naar een file of directory. De top van de hiërarchie zelf wordt trouwens '/' genoemd. Paden die niet met '/' beginnen heten '**relatief**' en beschrijven het pad naar een file vanaf de *huidige* directory. Net als in DOS is '..' een relatieve aanduiding die je in padnamen kunt gebruiken voor de directory *boven* je huidige directory, maar de '...' aanduiding van DOS om in één keer twee niveaus omhoog te gaan kent UNIX niet.

Vraag 1: Wat is het relatieve pad dat je kunt opgeven om in Linux in één keer twee niveaus omhoog te gaan in de directory hiërarchie?

Antwoord:

Ga terug naar je home directory met

```
cd ~
```

Hier gebruiken we de ~ afkorting voor je homedirectory. Als je de inhoud van de homedirectory van je buurman wilt zien en je kent zijn loginnaam, kan je proberen.

```
ls -a ~login
```

Het verschil tussen 'ls -a' en 'ls' is het wel of niet weergeven van 'verborgen' bestanden. De '-a' die je toevoegde aan het 'ls'-commando is een voorbeeld van een *optie*. Opties veranderen het gedrag van een commando (soms een beetje, soms veel). Het 'ls'-commando heeft flink wat opties.

Probeer 'ls' ook eens met deze opties:

```
ls -l      # de letter 'l'  
ls -1      # het cijfer '1'
```

Vaak, maar niet altijd, kun je opties combineren:

```
ls -al     # de letter 'l'  
ls -a -l   # werkt precies hetzelfde
```

Van de meeste commando's kun je een lijst met opties krijgen door '--help' mee te geven aan het commando:

```
ls --help
```

Vraag 2: Met welke optie kun je 'ls' zijn eigen versienummer laten uitprinten?

Antwoord:

3. Bestanden manipuleren

Om te oefenen met de commando's voor bestandsbeheer, hebben we een paar oefenfiles en -directories nodig. Maak in je home directory een directory aan met

```
mkdir experimentjes
```

en 'cd' naar die directory. Maak een tekstbestandje aan op de volgende, wat onhandige, manier:

```
cat > textje
```

vervolgens tik je twee regels onzin, en dan, op een lege regel, tik je Ctrl-d.

Nu moet er een bestandje zijn aangemaakt. Controleer dat met 'ls'.

Op de functie van dat groter-dan teken komen we zo terug. Dan verklaren we ook wat '**cat**' in dit geval deed. Maar om de functie van cat nu wat mysterieuzer te maken: je kunt '**cat textje**' gebruiken om de inhoud van je bestandje te bekijken.

Hier zijn nog een paar standaard bestandsbeheer commando's voor bestandsbeheer:

cp bestand 1 bestand2	bestand 1 kopiëren naar bestand2	(<i>copy</i>)
rm bestand	bestand verwijderen	(<i>remove</i>)
mv bestand directory	bestand naar directory verplaatsen	(<i>move</i>)
mv bestand1 bestand2	bestand1 hernoemen tot bestand2	

Let op dat 'mv' twee verschillende functies vervult. Maar als je erover nadenkt, is het niet helemaal onlogisch...


Als we een commando een naam van een file of directory meegeven, dan zeggen we dat die naam een *argument* is voor het commando. Argumenten zijn nodig om een programma meer te vertellen over *waarmee* het iets moet doen (welke file moet verwijderd worden), *hoe* het iets moet doen (in 'ls -a' is -a ook een argument), en soms zelfs *wat* het precies moet doen (bijvoorbeeld de --help die je 'ls' meegaf).

Tussen het commando en alle argumenten staat spaties. Als je een spatie wilt gebruiken *in* een argument, dan kan dat door aanhalingstekens te gebruiken:

```
mkdir "directory met spaties"
```

Gebruik deze commando's om in je 'experimentjes' directory de volgende directory-structuur na te bouwen:

```
experimentjes
|--dir1
|   |--textje
|   |--texttexttext
|   |--pipodeclownenmamalou
|
|--source
    |--myprog.c
    |--myprog.h
    |--myprog.o
    |--myprog.3
    |--myprog.readme
    |--makefile
```



Dit zijn files, geen directories

Dit zijn files, geen directories

waarbij de bestanden in 'dir1' en 'source' allen kopieën zijn van ~/experimentjes/textje (terwijl textje zelf niet meer in experimentjes staat). Neem de namen hierboven letterlijk over, want dan kunnen we daarna verder met ...

4. Globben, tabben en greppen

'Globben' is de term voor het gebruik van wildcards als '*' en '?' om meerdere bestanden in een keer te kunnen benoemen. Met 'tabben' bedoelen we het indrukken van de tab-toets. Het gebruik van '*' om meerdere willekeurige tekens in bestandsnamen te vangen kennen jullie uit DOS, net als het gebruik van '?' om een enkel willekeurig teken te vangen. UNIX heeft meer mogelijkheden op dit gebied. Greppen slaat op het gebruik van de command line applicatie Grep. Maar eerst gaan we tabben.

Tabben

'cd' naar de 'dir1' directory die je hebt aangemaakt. Bekijk 'pipodeclownenmamalou' met

```
cat pipodeclownenmamalou
```

Het resultaat mag je niet verrassen, maar om die lange bestandsnaam vaak te moeten intikken wordt gauw vermoeiend. Probeer dit eens:

```
cat p[tab]
```

Dat is dus handig. Nog een oefeningetje:

```
cat t[tab]t[tab]
```

De tab toets is handig om een bestandsnaam af te maken. De andere handige toets is 'pijlje-omhoog' om een vorige commando op te roepen. Probeer maar. Je kunt het commando dan nog wijzigen. Als je merkt dat je een commando herhaaldelijk uitvoert, kan dit je veel tikwerk schelen.

Globbering *?[^]

Ga naar de source directory, en doe

```
ls -l
```

Dit geeft extra informatie over de files. Nu gaan we globben:

```
ls -l my*
```

```
ls -l ma*
```

```
ls -l myprog.?
```

Dit kun je niet in Windows op de command prompt:

```
ls -l myprog.[ch]
```

```
ls -l myprog.[^ch]
```

```
ls -l myprog.[0-9]
```

```
ls -l myprog.[a-z]
```

Probeer nu eens voor jezelf te verwoorden wat die constructies betekenen. Voor optimaal begrip kan je de laatste nog een keer proberen met een '*' erachter, en het verschil te zien met de versie zonder '*'. Het zoeken op basis van complexe tekstpatronen zullen we veelvuldig doen in de bio-informatica.

Reguliere Expressies

De eerder genoemde complexe tekstpatronen staan bekend onder de naam "regular expressions" of in het Nederlands als "reguliere expressies". In plaats van het zoeken naar een letterlijke tekst stel je een patroon samen. Bijvoorbeeld: `a[gc][gta]` zoekt een codon `agg` of `agt` of `aga` enzovoorts. Voor bio-informatici zijn reguliere expressies handig om in DNA of proteïnes patronen terug te vinden. Voor UNIX systemen is het handig om bestanden terug te vinden.

Greppen

De command line applicatie 'grep' kan je gebruiken om te zoeken via de terminal. De afkorting staat voor Global Regular Expression Print. 'grep' zoekt in een bestand (of in standaard input als er geen bestand opgegeven is) naar tekstpatronen. Hiervoor kan je gebruik maken van regular expressions. Om meer te weten te komen over 'grep' kan je de man pagina's lezen:

man grep

Hierin wordt ook ingegaan op de regular expressions waar 'grep' mee om kan gaan.

Vraag 3: `[abcd]` betekent...

antwoord:

Vraag 4: `^[xyzq]` betekent...

antwoord:

Vraag 5: Zoek uit op wikipedia welke mogelijkheden er zijn met regular expressions.

antwoord:

5. De Manual Pages

De UNIX manual: 'man'

Tot nu toe hebben vooral dingen behandeld die ook in DOS gebruikt worden, met een paar verschilletjes. DOS lijkt ook op een uitgekleden UNIX. We willen laten zien dat UNIX op veel fronten krachtiger is dan DOS/Windows, *als* je tenminste bereid bent niet terug te schrikken van vaak cryptisch aandoende commando's.

Laten we maar gelijk in het diepe duiken:

`man ls`

Dit roept een programma op dat de manual page voor het 'ls' commando in beeld brengt. NIET HELEMAAL LEZEN, nog niet. Eerst leren hoe je bladert door de tekst. Dit zijn toetsen die je wilt weten:

spatiebalk	<i>scrollt een pagina naar beneden</i>
Enter	<i>scrollt een enkele regel naar beneden</i>
B	<i>scrollt een pagina naar boven</i>
Q	<i>Stoppen</i>
<i>/woord (slash /)</i>	<i>zoek eerstvolgend voorkomen van woord</i>
N	<i>zoek nog een voorkomen van het woord dat je met '/' zocht.</i>

Terwijl je bladert door de tekst, merk je dat de manual page heel formeel geschreven is, gedetailleerd en technisch. De meeste manual pages zijn op die manier geschreven en vermoeiend om te lezen. Naarmate je meer ervaring hebt met UNIX wordt het wat minder cryptisch, maar het blijft de kunst om te vinden wat je nodig hebt, en de rest te negeren. Het grote voordeel van manuals is dat ze, behalve exact en compleet, altijd in de buurt zijn.

Ga terug naar je home-directory.

`cd ~`

Vraag 6: Stel, om de één of andere reden wil je een lijst van alle bestanden in de héle subdirectory-hiërarchie onder je home-directory. Als je nu weet dat de UNIX-term voor 'de hele hiërarchie aflopen' 'recursive' is, welke optie van ls geeft je wat je wilt?

Antwoord:

Vraag 7: Hoe sorteer ik de uitvoer van ls op extensie? (uitproberen in de 'source directory')

Antwoord:

Onder DOS/Windows hebben files meestal een extensie van drie of vier letters waaraan het bestandstype af te lezen hoort te zijn. Onder UNIX zal je merken dat dat wel vaak gedaan wordt, maar lang niet altijd. Het is meer een gewoonte om het te doen voor sommige soorten bestanden, dan een verplichting. Het besturingssysteem 'doet niet' aan extensies, en de punt is geen speciaal karakter (behalve dus aan het begin van een naam). Vandaar dat om bijvoorbeeld alle bestanden te verwijderen 'rm *' voldoende is; de '*.*' van DOS is niet nodig.

We gaan de 'dir1' directory in 'experimentjes' verwijderen. Ga naar 'experimentjes'.

```
rm dir1
```

OK, wat werkt dan wel? UNIX heeft het commando 'rmdir' om een heel directory te verwijderen:

```
rmdir dir1
```

Toch kan je in UNIX een hele directory hiërarchie verwijderen met een enkel commando. Zoek het op in de man-page van 'rm', en verwijder 'dir1' met een commando van 10 letters (incl. spaties).

Vraag 8: Hoe verwijder ik 'dir1' met een commando van (in totaal) 10 tekens?

Antwoord:

Ook 'cd' kent parameters. Toch werkt 'man cd' niet. Probeer het.

Het kan zijn dat je toch een pagina krijgt, maar die gaat dan niet over het 'cd' commando dat we kennen, maar over een 'cd' in de programmeertaal TCL.

Het is tijd om kennis te maken met de *shell*.

De shell

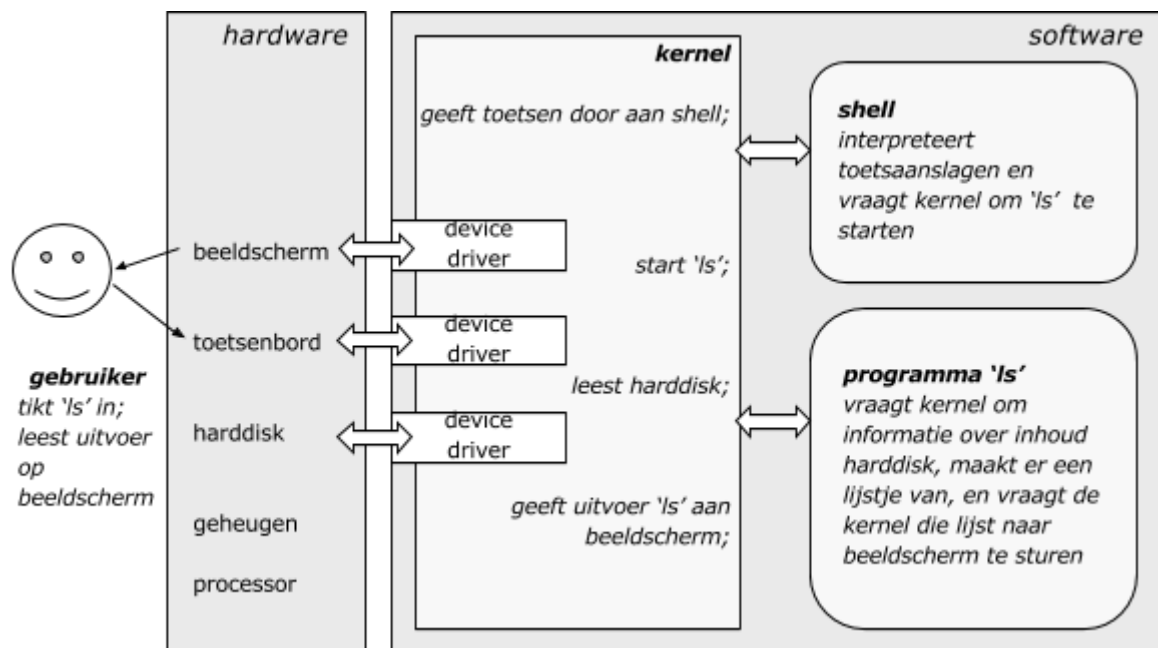
Wanneer je commando's aan UNIX aan het geven bent, ben je niet *direct* aan het praten met het hart van het besturingssysteem. Het hart van een UNIX-systeem, de '**kernel**', verstaat geen commando's. Het verstaat *system-calls* en *traps*, maar die kunnen alleen door programma's gegeven worden. De **shell** is het programma dat jouw toetsaanslagen vertaalt naar commando's, en commando's naar 'system-calls'. Dat is onder DOS niet anders, daar heet de shell COMMAND.COM, en de Explorer van Windows is ook een soort shell.

Alle shells hebben twee hoofdtaken: helpen met bestandsbeheer, en andere programma's starten (en stoppen). Ze delegeren veel aan andere programma's. Als je de shell vraagt om een lijst met de bestanden in de huidige directory door 'ls' te tikken, dan voert de shell dat commando niet zelf uit. Het gaat op zoek naar een programma dat 'ls' heet, en voert dat uit. Maar vanwege een technisch detail kan de shell het veranderen van de huidige directory niet overlaten aan een ander programma^{1*}. Dus doet-ie het zelf.

De shell die we onder Linux gebruiken heet '**bash**' en het is de slimste shell die er in de UNIX wereld bestaat. Er bestaan er meer; met name 'sh', 'ksh' en 'csh' zijn vrij bekend. Hoewel we tot nu toe nog weinig spectaculaire dingen hebben gedaan, hebben we bash al wel aan het werk gezet. Bash heeft de programma's 'ls', 'rm', 'man' etc. voor ons gestart. Maar bash heeft ook p[tab] voor ons vertaald naar 'pipodeclownenmamalou'. Het is bash dat globbing deed: 'ls -l myprog.[a-z]' werd vertaald naar 'ls -l myprog. c myprog.h myprog. o' *voordat* het 'ls' programma werd gestart.

We gaan de shell erg goed leren kennen...

^{1*} Waarom dat zo is, hoef je niet te weten. Maar als je het wilt weten: de werkdirectory wordt door de kernel *per proces* bijgehouden, en daarom kan een sub-proces nooit de werkdirectory van z'n *parent* wijzigen. Wanneer de shell een *ander* programma start, is dat andere programma automatisch een *subproces*. Vandaar dat de shell deze klus zelf moet doen.



Figuur 4: Schematische weergave Eigen werk auteur

Type op je scherm

```
man bash
```

Dit is één van de grootste manual pages die er zijn. Het 'cd' commando wordt beschreven in de buurt van regel 3600. Het is nu handiger om 'bash' zelf om hulp te vragen:

```
help
```

```
help cd
```

Als in UNIX documentatie een argument van een commando tussen vierkante haken [] staat, dan betekent dat dat je het weg mag laten. Maar nu we weten hoe het cd commando er uit kan zien, kunnen we gaan klooien.

Ter voorbereiding gaan we eerst even weg uit de home directory:

```
cd /var
```

Had je gezien dat je bij 'cd' *alles* weg kan laten? Probeer maar

```
cd
```

Vraag 9: Waar zit je nu?

Antwoord:

Is het je opgevallen dat aan het begin van elke man-page een 'NAME' regel staat, met daaronder de naam van het programma, en een korte beschrijving? Probeer anders

```
man man
```

voor een voorbeeld. In die korte beschrijvingen kan je zoeken. Stel, we hebben een tekst editor nodig. Dan kunnen we zoeken naar 'editor' in de 'NAME'-secties van alle man-pages:

```
man -k editor
```

De beschrijvingen zijn niet helder, maar we kunnen wel veelbelovende kandidaten uit de lijst uitproberen.

Info, de opvolger van man

Gebruikersvriendelijker dan 'man' is 'info'. Probeer maar:

```
info ls
```

'info' is een hypertext-programma, en de pagina's bevatten, net als www-pagina's, links naar andere info-pagina's. Dit keer kun je door de tekst bladeren met de gewone pijltjes-toetsen, en als je een stukje tekst tegen komt dat er zo uitzien: ** stukje tekst:*, dan kan je er met de cursor tussen gaan staan, en met de entertoets de link volgen. Stoppen met 'info' doe je met 'q'.

De auteur van de 'ls' info-pages heeft z'n best gedaan om wat overzichtelijker te zijn dan de man-page. Dat is lang niet altijd zo. Veel info-pages bevatten dezelfde tekst als de man-page over hetzelfde programma.

6. Tekst bewerken: VI

Hoewel 'info' vaak prettiger is om te gebruiken dan 'man', hebben we expres meer aandacht besteed aan 'man' dan aan 'info'. 'info' is namelijk een programma dat vooral op Linux machines aanwezig is. Andere UNIXen, die je op stage of op je werk tegen kan komen beschikken meestal niet over 'info'. Maar 'man' is *altijd* aanwezig, en daarom goed om in je UNIX-arsenaal te hebben.

Voor 'vi' geldt hetzelfde. Er zijn heel veel tekst editors voor UNIX. Sommige zijn krachtiger, veel zijn er vriendelijker. Maar 'vi' is er *altijd*. Daarom is het de moeite waard 'vi' een beetje te leren kennen.

Als project kiezen we ervoor een simpel programmaatje te schrijven. We noemen het 'lower'. Maar eerst gaan we oefenen.

vi oefening

De '~' regels zijn regels die nog niet bestaan in onze tekst. Nu is het hele scherm gevuld met dat soort regels, omdat onze tekst nog leeg is. Daar gaan we wat aan doen, maar lees even verder:

Het belangrijkste dat je moet weten van vi is dat vi in twee '**modes**' kan verkeren. In de '**insert**'-mode, zal vi je toetsaanslagen interpreteren als tekens die aan de tekst toegevoegd moeten worden. In '**command**'-mode worden je toetsaanslagen geïnterpreteerd als opdrachten aan vi zelf. Als je in insert-mode 'ZZ' tikt, verschijnt het in je tekst. Tik je 'ZZ' in in command-mode, dat verlaat je het programma.

Dit is de belangrijkste reden dat vi door sommigen vervelend wordt gevonden. Elke UNIX gebruiker kent de vergissing dat je je tekst begint te tikken terwijl je in command-mode zit, en in plaats van je tekst toe te voegen gaat vi allerlei gekke dingen doen in reactie op wat hij denkt dat commando's zijn. Sommigen kunnen zich daar aan ergeren, anderen kiezen ervoor om eraan te wennen.

'vi' is net opgestart, en zit in command-mode. Ga naar insert-mode met de 'i' toets:

i

Meestal verschijnt nu onderin beeld de tekst '-- INSERT --'. Prachtig, je zit in insert-mode.

Tik je naam maar in.

Spring weer terug naar command-mode met de Escape toets. Verlaat vi:

ZZ **vergeet niet de Shifttoets ingedrukt te houden**

Vi verdwijnt, en we kunnen weer commando's aan de shell geven. Verifieer dat er een bestand is aangemaakt met de naam 'oefening'.

We gaan aan ons programmaatje werken

vi lower.c

Hier is een lijstje vi-commando's die handig kunnen zijn (ze werken alleen in command-mode):

i	naar insert-mode
<Esc>	(in insert_mode) naar command-mode
h, j, k, l	cursor verplaatsen (links, onder, boven en rechts) De Linux versie van vi ondersteunt ook het gebruik van de pijltjes-toetsen (die werken zelfs in insert-mode). Maar niet alle versies van vi ondersteunen dit.
x	karakter onder cursor verwijderen
5x	vijf karakters verwijderen
dd	regel verwijderen (en in het clipboard zetten, dus 'knippen')
u	undo. Laatste handeling ongedaan maken. Dit is trouwens een functie die specifiek is voor de Linux versie van vi. Versies voor andere UNIXen hebben soms, maar niet altijd een undo. De undo van deze vi is ongelimiteerd, je kan het meerdere keren achter elkaar gebruiken.
8dd	acht regels verwijderen
yy	regel kopiëren naar clipboard
90yy	negentig regels kopiëren naar clipboard
p	inhoud van clipboard 'plakken' onder de huidige regel
34G	ga naar regel 34 (let op: hoofdletter G)
J	huidige regel samenvoegen met regel eronder.
:q!	vi verlaten zonder te bewaren
:wq	bestand bewaren en vi verlaten (net als ZZ)
:w	bestand bewaren
:w filenaam	bestand onder een andere naam opslaan
/woord	zoek naar woord voorwaarts

?woord	zoek naar woord achterwaarts
n	zoek naar volgend voorkomen
:help	roep helpscherm op, met uitleg over nog meer commando's

Tabel 4.1 vi -commando's

Gewapend met de bovenstaande tabel, maak het volgende programma exact na. Het is geschreven in C, de programmeertaal die is uitgevonden om UNIX mee te bouwen.

```
#include <stdio.h>
#include <string.h>

int main()
{ char c;

  c=getchar();
  while( c != EOF )
  {
    putchar( tolower(c) );
    c=getchar();
  }
  return 0;
}
```

Sla het bestand op, en verlaat vi.

Dit is een uiterst simpel C-programma dat alle hoofdletters in z'n invoer verandert in kleine letters. We gaan het verderop in de cursus gebruiken, maar het is niet erg als je niet snapt wat de code allemaal betekent. Probeer het gewoon precies over te nemen.

We moeten het bestand compileren om een uitvoerbaar programma te maken. Typ

make lower *niet 'lower.c'*

Er verschijnt een regel met 'cc lower.c -o lower'. Dat is de C-compiler die start. Als alles goed gaat, verschijnt kort daarna de prompt weer, zonder foutmeldingen.

Met een beetje pech verschijnen er wel foutmeldingen. Dat heb je de tekst niet helemaal exact overgenomen. Mooi. Dat betekent dat je met vi de tekst moet verbeteren. Let bij de foutmeldingen op de regelnummers die meteen achter de bestandsnaam vermeld worden. De fout zit meestal in die regel, of één erboven. Aarzel niet om je buurman/vrouw te vragen mee te kijken wat er niet klopt.

Als het compileren gelukt is heb je in je huidige directory een bestand staan dat 'lower' heet. Dat is het programma geworden. Start het maar op.

```
./lower # de ./ is nodig onder Red Hat Linux, vanwege de veiligheid.
```

Er lijkt niets te gebeuren, maar tik maar eens een regel in met wat hoofdletters erin.

Als je uitgespeeld bent, kun je op een lege regel Ctrl-D geven op 'lower' te stoppen. Wat dat betreft reageert 'lower' net als 'cat'. Straks zien we waarom. Probeer eens het volgende (vergelijk met wat we met cat deden):

```
lower > lowtextje # denk onder Red Hat Linux aan de ./
```

Typ weer een paar regels met hoofdletters. Dit keer verschijnt er geen uitvoer. Stop 'lower'. Ontdek dat er in je huidige directory een bestand is gemaakt met de naam 'lowtextje'. Bekijk het met vi. Dit was je tweede ervaring het omleiden van uitvoer:

7. Omleidingen en pijpleidingen

Invoer en uitvoer omleiden

Als we achter een commando-regel '> *bestandsnaam*' plakken, dat is dat het beste te begrijpen als dat de *uitvoer* van het programma word **omgeleid** van het scherm naar een bestand. Dit werkt voor bijna elk programma dat je onder UNIX kunt gebruiken.

Overtuig jezelf. Voer de volgende experimenten uit, en bekijk de veroorzaakte bestanden met het programma 'more':

```
pwd > pwdtextje
more pwdtextje
ls -l > lstextje
more lstextje
```

'more' is een standaard UNIX programma waarmee je tekstbestanden kunt bekijken. 'less' is een opgevoerde versie van 'more', die populair is onder Linux, maar niet vaak aanwezig is op andere UNIXen.

Eén van de fijnere eigenschappen van 'more' is dat, wanneer een tekst te groot is voor het beeldscherm, het even pauzeert. Met de spatiebalk, of de enter-toets kunnen we dan verder kijken, de 'b' toets scrollt een pagina terug. Als je aan het einde van het bestand bent gekomen stopt 'more', tussentijds stoppen doe je met 'q'.

```
man man > mantextje
more mantextje
```

Herhaal de oefening van de vorige pagina:

```
lower > lowtextje
```

Bekijk het bestand en merk op dat de vorige inhoud van lowtextje *zonder waarschuwing* is vervangen.

UNIX heeft sowieso veel minder dan Windows de neiging je te waarschuwen voor domme acties. Zo zal bijvoorbeeld 'rm -R *' zonder slag-of-stoot je hele directory wissen. Er is geen 'undo'. ('rm' heeft gelukkig de '-i' optie die het gebruik wat minder link maakt door voor alle bestanden die hij wil verwijderen eerst toestemming te vragen. Dat staat in de man-page)

Veiliger, en af en toe handig, is de mogelijkheid om de uitvoer van een programma *toe te voegen* aan een bestand:

```
lower >> lowtextje
```

Verifieer dat de vorige inhoud van lowtextje niet gewist is.

We kunnen ook de *invoer* van een programma omleiden.

Maak eerst, met 'vi' of 'cat' een bestandje aan met een paar regels die ook hoofdletters bevatten. Noem het bijvoorbeeld 'lowtext', en voer dan het volgende uit:

```
lower < lowtext
```

Wat zie je?

We kunnen ook nog *zowel* de uitvoer *als* de invoer omleiden. De file 'mantext' die we gemaakt hadden bevat genoeg hoofdletters:

```
lower < mantext > mantext.low
```

Bekijk het met 'more'.

Het is tijd voor wat achtergrond:

Een programma als 'lower' is te simpel om door te hebben dat het zijn invoer uit een file moet halen in plaats van het toetsenbord. Ook voor het omleiden van de uitvoer onderneemt 'lower' geen speciale actie. Hoeft ook niet.

Onder UNIX is elk draaiend programma (*proces*) aangesloten op een **standaard invoer** en een **standaard uitvoer**. Wat het programma betreft zijn dat gewoon bestanden. Het is het besturingssysteem dat ervoor zorgt dat de regels die je intikt, doorgegeven worden aan het programma via zijn standaard invoer alsof ze uit bestand komen. En meestal worden de tekens die een programma naar zijn standaard uitvoer schrijft doorgestuurd naar het beeldscherm (of in dit geval, via het netwerk naar jouw telnet venster).

Maar het besturingssysteem heeft er geen probleem mee om de standaard invoer of uitvoer aan te sluiten op een 'echt' bestand op schijf, en kan dat doen buiten medeweten van het betreffende proces om. Vandaar dat 'lower' geen verstand hoeft te hebben van het verschil tussen een toetsenbord en een file. Hij leest gewoon braaf de letters één-voor-één uit het ding waar de standaard invoer op is aangesloten. Het omleiden van de invoer of uitvoer van een programma gebeurt in een één-tweetje tussen de shell en de kernel voordat het programma wordt gestart.

Voer even de volgende drie commando's achter elkaar uit.

```
ls -l /usr/bin
ls -l /usr/bin > usrbin.txt
more usrbin.txt
```

In `/usr/bin` staat een groot deel van alle programma's die op een UNIX systeem zijn geïnstalleerd. Je hebt nu een lijst van die programma's, met details, in een tekstbestand.

Vraag 10: Op welke kolom is deze lijst gesorteerd?

Antwoord:

We gaan deze lijst sorteren om de grootte van het bestand (de vijfde kolom). Voer uit:

```
sort -n -k5 < usrbn.txt
```

De `-n` optie zorgt ervoor dat 'sort' *numeriek sorteert*, in plaats van alfabetisch (probeer het maar eens zonder de `-n`). De `-k5` optie zorgt ervoor we sorteren op de inhoud van de vijfde kolom (zoek maar op met 'man sort').

Pipeline

We hebben de uitvoer van 'ls -l' gesorteerd met het commando 'sort', maar daarvoor hebben we wel een bestandje op schijf moeten aanmaken: eerst de uitvoer van ls aansluiten op een bestand ('usrbin.txt'), en daarna dat bestand aansluiten op de invoer van 'sort'.
Waarom zouden we niet de uitvoer van 'ls' *direct* aansluiten op de invoer van 'sort'?

We bekijken even de directory `/etc`, waar veel configuratiebestanden van UNIX staan:

```
ls -l /etc | sort -n -k5
```

de verticale streep zit meestal in de buurt van de <enter>-toets

Nu is er dus *geen* bestand op harde schijf aangemaakt.

Dit heet een *pipeline*. De standaard uitvoer van het ene commando wordt dus via een pipeline direct doorgestuurd naar het andere commando, die de informatie ontvangt op z'n standaard invoer kanaal. Een heel handig commando om te gebruiken aan het eind van een pipeline is 'less'. Dat commando kan een grote hoeveelheid tekst in beeld brengen, pauzeren als het scherm vol is, en de gebruiker kan zoeken in de tekst (de toetsen zijn dezelfde als die je gebruikt om door man-pagina's te bladeren). Het 'cat' commando, dat je al gebruikt hebt, wordt veel gebruikt aan het begin van een pipeline om de inhoud van een file naar andere commando's te sturen.

Vraag 11: Maak een pipeline die, met gebruik van sort, 'cat' en 'less' de inhoud van usrbins.txt gesorteerd in beeld brengt.

Antwoord:

Bij het schrijven van een programma wil het nog wel eens voorkomen dat bij het uitvoeren van het programma de computer traag wordt. In dat geval kan je heel makkelijk de processen opvragen die het meest processor geheugen gebruiken met het commando 'top'. 'top' is een applicatie die op volgorde van meeste verbruik zijn processen sorteert en dit update.

Open een browser (bijvoorbeeld chrome) en zet iets aan dat relatief veel geheugen gebruikt, zoals Youtube. Voer nu het commando 'top' in in de command line en kijk wat er gebeurt. Je zal veel informatie zien over het proces en in de laatste kolom zie je 'command' staan, waar waarschijnlijk de browsernaam ergens bovenaan te zien zal zijn.

Probeer ook de volgende varianten uit en kijk weer wat er gebeurt:

```
top -u <jouwgebruikersnaam>
top | grep <jouwbrowser>
```

Zoals je ziet kan je dus ook grep gebruiken om een proces van een bepaalde applicatie op te halen, maar je ziet waarschijnlijk dat dit voor 'top' minder ideaal werkt met grep. Je kan ook processtatus opvragen met het commando 'ps'. 'ps' werkt iets anders dan 'top'. Waar 'top' zeer geschikt is om te kijken welke processen het meest actief zijn, is 'ps' geschikt om gedetailleerde informatie over allerlei processen op te vragen. 'ps' is zeer goed te combineren met grep.

Voer de volgende commando's in en kijk wat er gebeurt:

```
ps aux
ps aux | grep <jouwgebruikersnaam>
```

Vraag 12: Zoek met behulp van

```
ps --help of man ps
```

naar informatie over het gebruik van ps. Leg uit wat 'ps aux' doet.

Antwoord:

8. rwxr-xr-- en andere permissies

Sommige dingen mogen niet

Bekijk, met het 'less'-commando de inhoud van het bestand /etc/passwd:

```
less /etc/passwd
```

Het bestand 'passwd' in de directory '/etc' bevat de lijst met gebruikers die toegang hebben tot het systeem. Jij zou daar dus ook tussen moeten staan. Je ziet dat het bestand veel meer informatie bevat dan alleen je loginnaam. Elke regel bevatte een aantal velden met informatie, en tussen die velden staan dubbele punten (:). Het eerste veld is je login naam.

Bewerk /etc/passwd:

```
vi /etc/passwd
```

Zoek de regel op met jouw gegevens, en verander je login naam.

Bewaar het document in vi:

```
:w
```

Vraag 13: Wat is de reactie van het systeem (letterlijk) ?

Antwoord:

Verlaat vi:

```
:q!
```

Je ziet dat je /etc/passwd wel mag lezen, maar niet mag veranderen (schrijven). Dat is voor de veiligheid van het systeem, en van andermans data. UNIX is een multi-user systeem, waarop dus meerdere gebruikers hun bestanden bewaren. Het is niet de bedoeling dat iedereen jouw bestanden kan verwijderen of wijzigen.

Soms is het zelfs niet de bedoeling dat iedereen een bestand kan *lezen*. Het bestand waarin de wachtwoorden worden opgeslagen mag niet door iedereen gelezen worden. En niet altijd wil een systeembeheerder dat iedereen zomaar elk programma kan *uitvoeren* (het programma 'shutdown', bijvoorbeeld).

UNIX kent daarom 'permissies' die met één enkele letter worden aangeduid, dat zijn rechten om een bestand te lezen (*r*), ernaar te schrijven (*w*), en uit te voeren (*x*) als het om een programma gaat. Als eigenaar van een bestand mag je zelf weten welke permissies je toekent aan anderen, en kan je je bestanden zo geheim of onkwetsbaar maken als je wilt (hoewel de systeembeheerder altijd *alles* mag, dus helemaal geheim wordt je informatie nooit).

We gaan permissies bekijken:

```
ls -l /etc/passwd
```

de letter l, niet het cijfer 1

Vraag 14: wat zijn de eerste 10 letters van het antwoord?

Antwoord:

Bekijk, ter vergelijking, ook even de permissies van je eigen bestanden:

```
ls -l ~
```

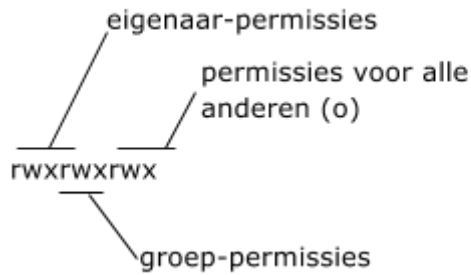
De eerste letter op een uitvoerregel van 'ls -l' zegt iets over het soort bestand (daar staat bijv. een 'd' als het bestand een directory is).

De volgende *negen* letters zijn de permissies voor dat bestand. In groepjes van drie staan de permissies aan (rwx), uit (---) of een combinatie daarvan (r--, voor alleen lezen, niet schrijven, niet uitvoeren). Het eerste groepje van drie bevatte de permissies die de eigenaar van het bestand *zichzelf* heeft gegeven.

Vraag 15: welke handeling (lezen, schrijven, uitvoeren) heeft de eigenaar van /etc/passwd zichzelf verboden?

Antwoord:

Het laatste groepje van drie permissies bevat de permissies die de eigenaar aan *alle andere gebruikers* geeft. Het groepje van drie permissies in het midden bevat de permissies die de *groepsleden* hebben. Bestanden horen namelijk niet alleen bij een bepaalde gebruiker (de eigenaar), maar ook bij een bepaalde *groep* gebruikers. Door permissies te aan of uit te zetten voor de groep, kun je sommige gebruikers meer rechten geven dan anderen. Zo kan je een document toegankelijk maken voor je projectgroep, en niet voor de rest van de gebruikers op het systeem.



Vraag 16: Als een bepaald tekstbestand leesbaar en schrijfbaar is voor de eigenaar, alleen leesbaar voor de groepsgenoten, en onleesbaar en onschrijfbaar voor alle andere gebruikers, wat zijn dan de negen permissie-letters voor dat bestand?

Antwoord:

De permissies veranderen CHMOD

Als je nog niet in je eigen home-directory zit, ga er dan naar toe.

```
cd ~
```

Maak een tekstbestand aan:

```
echo "pipo de clown" >> pipo.txt
```

en bekijk de inhoud van pipo.txt.

Vraag 17: Wat zijn de permissies voor pipo.txt, en mag de eigenaar (jij) ernaar schrijven?

Antwoord:

Voer het volgende commando uit:

```
chmod ugo-w pipo.txt
```

Vraag 18: Wat zijn nu de permissies voor pipo.txt?

Antwoord:

Probeer nog een keer naar pipo.txt te schrijven:

```
echo "pipo de clown" >> pipo.txt
```

het commando 'chmod' gebruik je om de permissies te wijzigen voor een bestand. Om die permissies te veranderen, moet je chmod vertellen *welke* permissies gewijzigd worden (r,w en/of x), of ze aangezet worden, of afgezet (+ of -), en voor *welke doelgroep* je dat wilt (eigenaar, groep, anderen). Doelgroepen worden met de letters 'u' (user, eigenaar), 'g' (group) en/of 'o' (others) weergegeven. Bijvoorbeeld:

```
chmod go+wx
```

zal de uitvoer- en de schrijf- en uitvoer permissies aanzetten voor groepsgenoten en anderen.

Vraag 19: Met welke commando's geef je jezelf lees- en schrijfrechten op pipo.txt, en zorg je ervoor dat groepsgenoten en anderen helemaal niets kunnen met pipo.txt?

Antwoord:

Permissie om te verwijderen

Zorg ervoor dat jijzelf helemaal geen rechten meer hebt op pipo.txt:

```
chmod ugo-rwx pipo.txt
```

Controleer dat je inderdaad niet kunt lezen schrijven naar dat bestand.

Probeer nu het bestand te verwijderen:

```
rm pipo.txt
```

(en antwoord met 'y' als het systeem aangeeft dat pipo.txt write protected is).

Je ziet dat je best bestanden kunt verwijderen, hoewel je er geen lees- of schrijfrechten op hebt. Dat lijkt merkwaardig, maar UNIX beschouwt het verwijderen van bestanden (en het maken van nieuwe) als schrijf-acties op de *directory* waarin je dat doet.

In UNIX kan een bestand namelijk best op twee plaatsen (directories) tegelijk aanwezig zijn. Als je het bestand uit de ene directory verwijdert, maar hij bestaat nog in de andere directory, dan blijft het bestand gewoon bestaan. Er is dan dus met het *bestand* niets gebeurd, maar wel met die ene directory waaruit je het bestand verwijderde.

Maak een test-directory aan:

```
mkdir testdir
```

En in die directory een bestandje:

```
cd testdir  
echo "pipo" >> pipo.txt
```

Bekijk de permissies op testdir:

```
cd ..  
ls -l testdir
```

Haal nu *alle* schrijfrechten weg van testdir:

```
chmod ugo-w testdir
```

En probeer nu pipo.txt te verwijderen:

```
cd testdir  
rm pipo.txt
```

Nu je geen schrijfrechten meer hebt op de directory mag je dus geen bestanden meer verwijderen uit die directory. Je kunt ook geen nieuwe bestanden meer aanmaken in deze directory.

Probeer maar (zorg ervoor dat testdir je huidige directory is):

```
mkdir nog_een_directory  
echo "pipo" >> nog_een_pipo.txt
```

9. Shell scripts

Commando's combineren

Soms zijn bepaalde *combinaties* van commando's handig. De volgende commando's geven een systeembeheerder wat informatie over de machine:

who	<i># toont de gebruikers die zijn ingelogd</i>
free	<i># toont informatie over het geheugengebruik in de machine</i>
df	<i># toont informatie over de harddisks in de machine</i>

Stel dat een systeembeheerder dit drietal commando's in één keer wil geven. Dan kan hij ze zo samenstellen:

```
who; free; df
```

Maar als hij deze combinatie heel vaak wil gebruiken, is het handig om de commando's *op te slaan* in een tekstbestand.

Gebruik 'vi' om een tekstbestand te maken (dat 'sysinfo' heet), met de volgende inhoud:

sysinfo:

```
who
free
df
```

Nu kan je deze commando's uitvoeren door de shell 'sh' te vragen de file te lezen en uit te voeren:

```
sh sysinfo
```

Je hebt nu je eerste *shellscript* gemaakt. Misschien heb je onder Windows wel eens een 'batch file' gemaakt of gezien; dit is net zoiets.

Een shellscript is een tekstbestand dat door de shell (het programma dat jouw commando's leest, en dan goede programma's opstart) gelezen en uitgevoerd wordt.

In shellscripts kan je serieuze programmatuur schrijven. Door variabelen te gebruiken, en commando's slim samen te stellen kunt je met shellscripts hele handige en flexibele hulpmiddelen schrijven.

Shell variabelen

Voer de volgende commando's één voor één uit:

```
A=3          # let op: géén spaties rondom de '='
```

```
echo A
```

```
echo $A
```

Het commando 'echo' zal de argumenten die je meegeeft afdrukken op het scherm (eigenlijk z'n standaard uitvoer).

Maar elke keer als je een \$-teken voor een woord zet (in dit geval de letter 'A'), dan denkt de shell dat het woord moet worden opgevat als een variabele, en moet *vervangen* worden door de waarde van die variabele.

Let erop dat we geen dollartekens gebruiken om een variabele een waarde te geven.

Voer de volgende commando's één voor één uit:

```
echo Hallo $naam, hoe gaat het.
```

```
naam="Pipo de Clown"          # vergeet de "" niet
```

```
echo Hallo $naam, hoe gaat het.    # precies hetzelfde als hierboven
```

Je ziet dat als en nog niets in een variabele staat, er ook niets wordt weergegeven als de variabele gebruikt wordt.

De aanhalingstekens waren nodig om spaties te kunnen gebruiken in de waarde van 'naam'.

Voer de volgende commando's één voor één uit:

```
a=3
```

```
b=4
```

```
c="$a + $b"
```

```
echo $c
```

Vraag 20: Wat zou je zeggen dat het datatype is van shellvariabelen: integers of strings?

Antwoord:

Je kunt de shell best laten rekenen. Vervang de aanhalingstekens door '\$[' en ']':

```
c=$(( $a + $b ))
```

```
echo $c
```


Variabelen in een script

We gaan een scriptje maken dat een directory archiveert. Eerst moet de complete inhoud van de directory samengevoegd worden tot één bestand. Dat bestand wordt dan gecomprimeerd, en de gecomprimeerde data wordt verplaatst naar een directory die 'archief' heet. Dan wordt de oorspronkelijk directory gewist.

Maak alvast de directory 'archief' aan, en kopieer de directory 'experimentjes', die we gaan gebruiken om mee te testen:

```
mkdir archief
cp -r experimentjes archtestdir
```

Gebruik 'vi' om een tekstbestand te maken (dat 'arch' heet), met de volgende inhoud:

arch:

```
dir=archtestdir          # alles achter '#' is commentaar
tar cvf $dir.tar $dir    # archtestdir.tar bevat alle samengevoegde
                          # bestanden
gzip $dir.tar            # comprimeer tot archtestdir.tar.gz
mv $dir.tar.gz archief   # verplaatsen
rm -r $dir               # verwijder archtestdir (en inhoud)
```

Probeer het uit:

```
sh arch
```

en test het succes met de volgende commando's

```
gunzip archief/archtestdir.tar.gz    # decomprimeren
tar xvf archief/archtestdir.tar      # uitpakken
```

Het script dat we gemaakt hebben is onhandiger dan nodig. Nu kan het alleen maar de directory 'archtestdir' archiveren.

Als we het willen aanpassen, zodat we 'arch' een naam kunnen opgeven, en dat 'arch' dan de directory met die naam archiveert, dan gebruiken we de variabele \$1. In een shell-script bevat \$1 de waarde van het eerste argument dat je meegaf toen je het script aanriep (\$2 het tweede argument, enz.).

Verwijder de eerste regel uit het 'arch'-script, en vervang overal de variabelenaam '\$dir' door '\$1':

Vervang de eerste regel uit 'arch' door de volgende:

arch:

```
tar cvf $1.tar $1          # $1.tar bevat alle samengevoegde
                             # bestanden
gzip $dir.tar              # comprimeer tot archtestdir.tar.gz
mv $dir.tar.gz archief     # verplaatsen
rm -r $dir                 # verwijder archtestdir (en inhoud)
```

en test het:

```
cp -r experimentjes tweededir
sh arch tweededir
ls -l archief
```

Tot nu toe hebben we ons script steeds aangeroepen met 'sh arch'. Het kan handiger door je script execute-permissie te geven. Dan kun je het script uitvoeren alsof het een normaal commando is. De shell heeft dan zelf wel door dat het om een script gaat.

Voer uit:

```
chmod u+x arch
cp -r experimentjes derdedir
./arch derdedir          # vergeet de './' niet; dit is nodig om veiligheidsredenen.
ls -l archief
```

10. File Transfer Protocol

Het file transfer protocol (ftp) is een krachtige manier om bestanden op een server te plaatsen en er bestanden vandaan te halen. Zeker wanneer grote databestanden heen en weer geschoven worden is het veel sneller dan over HTTP. HTTP is bedoelt voor het ophalen van Hypertext bestanden, FTP is juist bedoelt voor het plaatsen van bestanden op de server of het ophalen van grote bestanden.

Een browser kan overigens wel overweg met het ftp protocol. Probeer maar eens <ftp://ftp.ensembl.org/> in te tikken in een browser. Je kunt nu alle bestanden downloaden. Er zijn ook speciale ftp cliënt om te werken met ftp zodat je ook bestanden kunt uploaden.

```

Connected to extftp2.sanger.ac.uk.
220-ftp.ensembl.org NcFTPd Server (free educational license) ready.
220-Ensembl Project FTP server
220-
220-Problems after login? Try using '-' as the first character of your
220-password.
220
User (extftp2.sanger.ac.uk:(none)): anonymous
331 Guest login ok, send your complete e-mail address as password.
Password:
230-You are user #19 of 150 simultaneous users allowed.
230-
230- Welcome to the Ensembl anonymous FTP site
230- ~~~~~
230-
230-   DNA Sequence, Quality and Trace Data
230-   -----
230-   Users of the material provided here in advance of submission to
230-   public databases should read the Data Release Policy
230-   and Guidelines and Conditions on use of Data in the respective data
230-   directory.
230-
230-   Please report any unusual problems you may have with this server
230-   via e-mail to webmaster@ensembl.org
230-
230-   Problems? See our FTP Frequently Asked Questions:
230-
230-       http://www.sanger.ac.uk/Info/IT/ftp-faq.shtml
230-
230-   All connections and transfers are logged
230-*****
230-*****      FTP Server software upgrade      *****
230-***** The FTP Server software has been upgraded. *****
230-***** Please report any anomalies to          *****
230-*****      ftpadmin@sanger.ac.uk              *****
230-*****
230-*****      Issues with ncftp client resolved *****
230-*****
230-
230-Logged in anonymously.
ftp> ls
200 PORT command successful.
150 Opening ASCII mode data connection for /bin/ls.
bin
dev
etc
lib
ls-lR.Z
pub
quota.group
quota.user
226 Listing completed.

```

In bovenstaand voorbeeld wordt anoniem aangelogd aan de ftp server van Ensembl. Hier kunnen we biologische databestanden downloaden. Als bio-informaticus zul je vaak gebruik maken van ftp. Niet altijd is er een speciale ftp cliënt beschikbaar. Probeer daarom de basis van de ftp commando's te leren.

O.S. Prompt> ftp ftp.ensembl.org

Login met **anonymous**, dit is een veel gebruikte manier om anoniem aan een ftp server te connecten. Wel wordt soms daarna gevraagd om een e-mail adres.

```
ftp> ls
ftp> pwd
ftp> cd /pub
ftp> cd current_saccharomyces_cerevisiae
```

Bekijk nu eens in welke directory je zit.

```
ftp> cd data/fasta/dna
```

Tot nu toe lijkt alles erg veel op Linux/UNIX. Het ftp protocol is dan ook afkomstig uit de UNIX wereld.

Nu gaan we ftp specifieke commando's geven om bestanden binnen te halen.

```
ftp> bin
ftp> get <filename>
ftp> mget *
```

Met get haal je 1 gespecificeerd bestand op van de server. Met mget (multiple get) kun je meerdere bestanden gelijktijdig overhalen. Je mag dan gebruik maken van wildcards als *. De bin die je opgeeft zorgt er voor dat je bestanden als binaries worden overgehaald en niet als tekstbestanden.

Waar belanden deze bestanden? Standaard komen de opgehaalde bestanden terecht in de directory waar je de ftp cliënt startte. Zou je willen wisselen op je lokale PC dan kan dit met **lcd** (local change directory).

Het plaatsen van bestanden op de server kan in ons geval niet. Als anonymous gebruiker hebben we te lage rechten om files te uploaden. Maar zouden we die rechten wel hebben dan kunnen we gebruik maken van:

```
ftp> mput *
```

Tip: let altijd op of het formaat van uploaden/downloaden juist is. Dat wil zeggen tekstbestanden als ASCII en alle anderen als binaries.

Vraag 21: Wanneer heeft de laatste upload plaatsgevonden van het genoom van het bakkersgist?

Antwoord:

11. En nu verder

De basis van Linux is gelegd, maar vaak zul je net iets meer willen weten over Linux. Met het commando 'man en info' heb je gezien dat je van de commando's meer te weten komt.

Als bio-informaticus hoef je geen specialist te zijn op het gebied van Linux. Wel dien je in staat te zijn om eenvoudige commando's uit te voeren.