Unix

Informatica werktuigen

Inhoudsopgave

In deze oefenzitting behandelen we het alternatief besturingssysteem Linux, dat deel uitmaakt van de Unix-familie. Er valt veel te vertellen over dit besturingssysteem, te veel om allemaal te kunnen behandelen in twee uur. Daarom beperken we ons tot die kleine fractie aan functionaliteit waarvan het het meest waarschijnlijk is dat het van pas zal kunnen komen tijdens jullie studies.

> Een kleine nota vooraf: zoals met de meeste dingen in de informatica is leren werken met Linux grotendeels een kwestie van autodidactiek. De technologie gaat razendsnel vooruit en als goede informaticus dien je jezelf up-to-date te houden met de laatste ontwikkelingen in je specialiteit. Het gaat er dus niet om hoe goed je met een bepaalde library/programmeertaal/besturingssysteem/... kan omgaan, het gaat erom dat je je plan leert trekken wanneer je geconfronteerd wordt met iets nieuws, dat je er op autonome wijze mee leert werken door de documentatie te lezen, door rond te neuzen op internetfora, door te experimenten, etc.
>
> In principe zouden we je dus gewoon enkele oefeningen kunnen voorschotelen en jullie aan je lot overlaten. Dit is echter niet wat we zullen doen, omwille van de inefficiëntie van deze aanpak. De bestaansreden van deze opgave is om jullie een kickstart te geven: het moet je snel wat basiskennis meegeven waarop je dan later, op je eentje, voort kan bouwen. Een onontbeerlijk stuk gereedschap hierbij zijn de man-pages (kort voor *manual pages*), die documentatie bevatten over de (meeste) command-line commando's in Linux. Deze kan je raadplegen door op de command-line `man [commandonaam]` in te voeren. De documentatie verlaten doe je met `q`.

Voor je begint aan deze oefenzitting vragen we je om eerst nog enkele stappen te doorlopen, die nodig zijn om het groepswerk dat jullie in het kader van deze cursus zullen maken tot een goed einde te brengen. Deze staan uitgelegd op **Toledo** onder de sectie **Opdrachten → Aanmaken SSH Key** . Dit is belangrijk voor het groepswerk, dus **vergeet dit niet te doen!**

1 De command-line interface

Hoewel gebruikers doorgaans beroep doen op de grafische interface van programma's, kan het voordelig zijn ook te kunnen werken met de command-line interface (CLI):

- Bepaalde taken zijn veel sneller uit te voeren via de CLI dan met een GUI.
- De CLI biedt doorgaans een mini-programmeertaal waardoor het gemakkelijk is bepaalde taken te automatiseren: zo kan men werken met variabelen, lussen, de in- en uitvoer van programma's omleiden,...
- Soms is de CLI de enige interface ter beschikking.

In Linux wordt de command-line vaak gebruikt om snel en optimaal het besturingssysteem aan te spreken, en biedt het soms opties die niet via grafische interface kunnen worden aangeboden.

1.1 Eenvoudige commando's

Open de terminal (in het startmenu). Je krijgt dan een prompt te zien die er doorgaans als volgt uitziet:

```
<hostnaam>$
```

Waar `hostnaam` de naam is van de pc waarop je ingelogd bent. Er is ook sprake van een huidige directory (de *working directory*): initieel is dit je eigen home directory, een stuk opslagruimte waar enkel jij toegang tot hebt. Als je wil, kan je de prompt wijzigen om de huidige directory weer te geven:

```
<hostnaam>$ PS1='\w $ '
```

Natuurlijk moet je het gedeelte `<hostnaam>$` hierbij niet invoegen. Merk ook op dat er geen spaties mogen staan rond het gelijkheidsteken!

De `\w` staat hier voor working directory,[1] waardoor de prompt verandert naar

```
~ $
```

waarbij `~` staat voor je home directory. Dit teken kan je dan ook altijd gebruiken om naar een bepaald pad te verwijzen binnen je eigen thuisdirectory, zoals bv. `~/Pictures` dan staat voor je persoonlijke `Pictures` subdirectory. De wijziging van de prompt is slechts tijdelijk[2]: je kan deze herstellen door de terminal te verlaten met `exit` en vervolgens een nieuwe terminal op te starten.

In de rest van de opgave gebruiken we wel telkens de gewijzigde prompt om duidelijk te maken in welke directory je je moet bevinden bij het uitvoeren van bepaalde commando's. Commando's die in de opgave worden geïllustreerd worden vaak voorafgegaan door deze prompt. Wanneer jullie deze commando's zelf willen ingeven, moeten jullie dus enkel invoeren wat er **na** het `$`-teken komt!

Je kan de inhoud bekijken van je thuisdirectory met `ls`:[3] (*list directory contents*)

```
~ $ ls
```

(Dus de `~ $` niet intypen!) Normaalgezien zijn er reeds een aantal directories aanwezig:

```
Desktop
Pictures
Templates
Documents
...
```

[1] Meer opties over het wijzigen van de prompt kan je vinden op `http://tldp.org/HOWTO/Bash-Prompt-HOWTO/bash-prompt-escape-sequences.html` maar kan je ook vinden in de man page voor bash (`man bash`).

[2] Om je prompt permanent te wijzigen, moet je de `PS1=...` lijn plaatsen in het `.bash_profile` bestand in je home directory.

[3] Het `~ $` gedeelte is in dit geval dus de prompt, dit hoeft je dus niet over te typen.

Je kan deze directories bezoeken door middel van het `cd` commando (*change directory*):

```
~ $ cd Documents
~/Documents $ ls
[inhoud van je Documents directory, vermoedelijk leeg]
```

Let op: in tegenstelling tot Windows is Linux hoofdlettergevoelig: `cd documents` zal dus leiden tot een foutmelding. Je kan om typwerk te besparen ook gebruik maken van de autocomplete feature: als je `cd Doc` intypt, gevolgd door de tab-toets, zal dit automatisch vervolledigd worden tot `cd Documents` (voor zover er geen andere conflicterende mogelijkheden zijn).

Je kan terugkeren naar je thuisdirectory op de volgende manieren:[4]

```
cd ..        # terug naar de parent directory
cd ~         # rechtstreeks naar de home directory
cd           # geen argument = terug naar homedir
```

Je thuisdirectory zit ergens in een grotere boomstructuur. Waar dit exact is kom je te weten met het volgende commando:

```
~ $ pwd
```

hetgeen staat voor *print working directory*. Als antwoord krijg je

```
/home/rXXXXXXX
```

waar `rXXXXXXX` staat voor je r-nummer.

Maak nu een nieuwe directory aan in je huidige directory door middel van het volgende commando:

```
~ $ mkdir werkmap
```

Hierbij zal je een nieuwe directory aanmaken op het pad `/home/rXXXXXXX/werkmap`. Ga vervolgens in deze map door middel van het `cd`-commando.

Er zijn tal van commando's om bestanden en directories te manipuleren. We kunnen ze moeilijk allemaal apart bespreken, maar hier alvast een selectie van de meest interessante:

- `cp [from] [to]`: maakt een kopie van een bestand
- `cp -r [from] [to]`: recursief kopiëren, voor directories
- `rm [naam]`: verwijdert bestand (pas wel op, weg is weg onder Linux)
- `touch [naam]` maakt een nieuw, leeg bestand aan onder de huidige werkdirectory met de gegeven naam.
- `mv [from] [to]`: verplaatst een bestand of directory (kan gebruikt worden om te renamen)
- `du`: disk usage
- `mkdir [dir]`: aanmaken directory
- `find -name [naam]`: recursief zoeken naar een bestand of directory
- `cat [bestandsnaam]` drukt de inhoud van het bestand af
- `echo [tekst]` print `tekst` af

[4] Wat na `# volgt is commentaar en hoeft dus niet ingevoerd te worden.`

Je kan bij de meeste van deze commando's gebruik maken van wildcards, deze maken het mogelijk om een patroon voor te stellen voor bestandsnamen. Stel dat je alle bestanden met extensie[5] `.txt` wil verwijderen, dan kan dit met `rm *.txt`: de asterisk staat voor "nul of meer karakters", waardoor `*.txt` de verzameling bestandsnamen beschrijft die eindigen op `.txt`.[6]

Nu je een opsomming van enkele commando's hebt gekregen, kan je deze kennis ook onmiddellijk in de praktijk omzetten:

- Maak in je pas aangemaakte directory een nieuw, leeg bestand aan met de naam `nieuw.txt`.
- Kopieer nu dit laatste bestand naar `nieuw2.txt` en controleer of het bestand ook daadwerkelijk (1) in de directory staat en (2) geen inhoud heeft.
- Verwijder nu alle bestanden in `werkmap`.

Na deze kleine oefeningen kan je ook de directory `werkmap` die je zojuist maakte terug verwijderen. Ga hiervoor eerst terug naar je thuisdirectory. Vervolgens kan je de directory door middel van het `rm`-commando.

```
~ $ rm werkmap
```

Wanneer je dit probeert, krijg je wellicht de foutmelding

```
rm: cannot remove ‘werkmap’: Is a directory
```

Ga in de `man`-pagina van `rm` na hoe je dit commando toch kan gebruiken om directories te verwijderen door middel van een optie toe te voegen bij het commando.

Het kan de moeite waard zijn om enkele belangrijke directories in het bestandssysteem te vermelden (zie `man hier` voor een volledigere overzicht):

- `/media`: hier vind je doorgaans CD's, DVD's, USB-sticks, ... terug. Als je bijvoorbeeld een CD in de lezer plaatst, zal je de bestanden ervan vinden in `/media/cdrom0`.
- `/home`: zoals reeds vermeld, hier bevinden zich alle thuisdirectories.
- `/proc`: een directory met virtuele bestanden die allerlei informatie bevatten over het systeem en de actieve processen.

1.2 Streams

Een stream is een eenvoudige abstractie die een rij opeenvolgende stukjes data voorstelt. Men kan twee types onderscheiden: een source (of input stream) is een stream waaruit men gegevens kan lezen, en een sink (of output stream) is een stream waar men naartoe kan schrijven.

Als een programma opgestart wordt, krijgt deze standaard drie streams van het besturingssysteem:

- stdin: de standaard invoer;
- stdout: de standaard uitvoer;
- stderr: de standaard foutuitvoer.

Standaard wordt stdin gebonden aan de toetsenbordinvoer (dus als een programma leest uit stdin zal deze ontvangen wat de gebruiker intypt) en worden stdout en stderr geleid naar de terminal (m.a.w. het wordt gewoon afgeprint). De CLI laat ons toe om deze streams te manipuleren: we kunnen de stdout van één programma omleiden naar de stdin van een ander om zo de gegevens te filteren, te sorteren, etc. Dit omleiden gebeurt met de pipe operator `|`.

[5] In feite kent Linux geen extensies en is het punt een teken zoals een ander. Wel, ok, niet helemaal: als een bestandsnaam immers met een punt begint, wordt het beschouwd als een *hidden file*, m.a.w. het bestand zal niet getoond worden bij een gewone `ls`-opdracht. Om de verborgen bestanden toch te zien krijgen, kan je `ls -a` gebruiken.

[6] Zie `http://www.tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm` of `man bash` voor meer mogelijkheden.

Stel dat we willen weten welke directory het meeste plaats inneemt omdat we dichtbij onze quota[7] limiet zitten. De rauwe informatie over plaatsgebruik bekomen we met `du` (*disk usage*). Vervelend is dat standaard de grootte van een parent directory ook de groottes van alle subdirectories meetelt, waardoor gewoon de directory aan de top automatisch de grootste is. Kijken we naar de documentatie (de man pages) van `du`, zien we dat er een flag `-S` is die ervoor zorgt dat de grootte van een directory gelijkgesteld wordt aan de som van de grootte van de bestanden erin, zonder de subdirectories erbij te tellen.

De door `du` geproduceerde uitvoer is echter niet gesorteerd. Om dit te verhelpen, kunnen we het programma `sort` gebruiken dat zijn stdin invoer sorteert en het resultaat naar stdout wegschrijft. Vermits sorteren op veel manieren kan gebeuren, moeten we `sort` expliciet vermelden dat we wensen *getallen* te sorteren (in tegenstelling tot het alfabetisch sorteren van strings). Dit gebeurt met de `-n` flag (*numeric sort*). Standaard sorteert `sort` de data in stijgende volgorde, dit kunnen we omkeren met `-r` (*reverse*).

Er zijn mogelijk heel wat directories die afgeprint zullen worden, en we zijn enkel geïnteresseerd in de grootste. Daarom doen we beroep op `head`, dat standaard enkel de eerste tien binnenkomende regels tekst doorlaat naar de uitvoerstream.

We combineren deze drie programma's als volgt:

```
cd      # eerst terug naar thuisdirectory
du -S | sort -n -r | head
```

De pipe neemt dus de stdout van de linkeroperand en verbindt deze met de stdin van de rechteroperand. Een andere operator is `>`, die de stdout-uitvoer van het linkerproces wegschrijft naar een bestand waarvan de naam rechts van de operator staat:

```
du -S | sort -n -r | head > result.txt
```

Dit commando schrijft de top 10 grootste directories naar het bestand `result.txt`. Zoek nu op hoe je ook *enkel* de grootste directory te zien krijgt i.p.v. de top 10.

Ziehier een kort overzicht van programma's die van pas kunnen komen:

- `cat [bestand]` leest het bestand in en stuurt het naar stdout.
- `sort` sorteert alfabetisch. De `-n` flag interpreteert de strings als getallen. De `-r` flag keert de volgorde om van het resultaat (m.a.w. sorteren gebeurt in dalende volgorde).
- `grep [pattern]` laat enkel lijnen tekst door die het gegeven patroon (regular expression) bevatten.
- `wc -c` telt het aantal bytes, `wc -m` het aantal tekens, `wc -w` het aantal woorden, `wc -l` het aantal lijnen.
- `more` toont de uitvoer stuksgewijs: na elk scherm tekst wordt gewacht op een signaal van de gebruiker om de rest af te printen.
- `less` is een krachtigere versie van `more` doordat het o.a. ook toelaat terug naar omhoog te scrollen en te zoeken naar strings.
- `head -n [N]` laat enkel de eerste `N` lijnen door.
- `tail -n [N]` laat enkel de laatste `N` lijnen door.

Tijd voor enkele eenvoudige oefeningen op pipes:

- Zoek op welke processor in je machine steekt. Het (virtueel) bestand `/proc/cpuinfo` bevat hiervoor een lijn `model name : ....` Zoek eveneens met hoeveel RAM-geheugen de pc uitgerust is: doorzoek hiervoor het bestand `/proc/meminfo` en zoek naar `MemTotal`.
- Tel het aantal processors die werden opgenomen in het `/proc/cpuinfo` bestand. Dit kan je doen door te zoeken naar `processor` en de command-line hiervan de resultaten te laten tellen.

[7] Bekijk je quota met `quota`.

Extra oefening: een webpagina aanpassen

Nu jullie kennis hebben gemaakt met verschillende commando's, gaan we dit in de praktijk omzetten om een webpagina offline op een efficiënte manier aan te passen.

Hiervoor hebben jullie nog twee extra commando's nodig:

- `wget [URL]` kan gebruikt worden om via de commandline een bestand te downloaden.
- `sed [pattern] [stream]` kan worden gebruikt om een meegegeven bestand of andere sink-stroom aan te passen aan het gegeven patroon.

De patterns die gebruikt worden voor het `sed`-commando kunnen sterk uiteenlopen. Een pattern zal aangeven dat het commando een bepaalde bewerking moet uitvoeren op de delen van de invoerstroom die ermee overeenkomen. Probeer bijvoorbeeld het commando:

```
echo abba | sed s/a/b/g
```

Dit commando zal `bbbb` teruggeven. Merk op dat de uitvoer van `echo abba` hier als invoerstroom voor het `sed`-commando wordt gebruikt. Het pattern `s/a/b/g` geeft aan dat voor deze invoer karakters *a* moeten worden vervangen (aangegeven door de *s* van *substitute*) door het karakter *b*. De *g* in het pattern geeft aan dat dit moet gebeuren voor **alle** karakters *a*. Meer over het `sed` commando kan je vinden op de `man`-pagina of online[8].

Download nu met behulp van `wget` de webpagina `http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-3.html`. Binnen je huidige werkdirectory zal de HTML-broncode van deze webpagina hiermee worden gedownload.

Wanneer je de inhoud van dit bestand bekijkt (bijvoorbeeld met behulp van `less`) zal je de inhoud van deze webpagina zien, gestructureerd volgens enkele HTML-commando's. De precieze betekenis van deze commando's hoef je niet te kennen[9], maar je zal onder andere instructies zoals `<HEAD>`, `<BODY>`, `<A>` en `<P>` terugvinden in dit document, die iets zeggen over hoe de inhoud van de webpagina moet worden gestructureerd. De meeste van dergelijke instructies zijn van de vorm `<INSTRUCTIE>INHOUD</INSTRUCTIE>`, wat aangeeft op welke inhoud ze betrekking hebben. Eén van de instructies in het document is het `<H2>`-commando, dat de titels in het document structureert (dit kan je ook zien wanneer je de webpagina met de browser opent en de broncode ernaast legt). Er bestaan verschillende soorten titels in HTML, o.a. `H1`, `H2`, `H3`, ... Die aangeven of het over een titel, dan wel een ondertitel (enzovoort) gaat en zo het uitzicht van de webpagina ook kunnen beïnvloeden.

Wij willen nu graag alle `H2` vervangen door de `H1` equivalent in de zopas gedownloade pagina, zodat alle ondertitels in het document titels worden. Voer hiervoor twee opdrachten uit:

1. Tel eerst door middel van de commando's die we jullie hebben toegereikt om hoeveel ondertitels van de vorm `H2` het gaat.
2. Maak nu gebruik van de door ons aangeboden commando's om de broncode voor een nieuwe webpagina *nieuw.html* weg te schrijven die alle `H2` instructies door `H1` instructies vervangt.[10] Maak hierbij zoveel mogelijk gebruik van streams.

2 Git

2.1 Inleiding

Tijdens het groepswerk zullen jullie gebruik moeten maken Git, een *version control system (VCS)*. Een VCS laat het toe eenvoudig de veranderingen van bestanden doorheen de tijd bij te houden. Meestal integreert

[8] Meer voorbeelden van patterns voor `sed` kan je vinden op `http://www.tutorialspoint.com/unix/unix-regular-expressions.htm`.

[9] Wil je toch meer leren over HTML, dan bestaan er op het web talloze cursussen hierover. Een leuk beginpunt is `http://www.w3schools.com/html/`.

[10] Hint: de `>`-operator kan handig van pas komen!

een VCS ook een manier om de bestanden – inclusief hun geschiedenis – te delen met andere mensen. Dit zorgt ervoor dat een VCS een onmisbaar hulpmiddel is tijdens het ontwikkelen van software.

Buiten Git zijn er nog veel andere VCS'en – een paar van de meer bekende zijn Subversion en CVS. Onder andere door diensten als GitHub is Git de laatste jaren echter ongezien populair geworden, wat ervoor zorgt dat er online zeer veel documentatie over te vinden is. Jullie worden aangemoedigd van deze documentatie gebruik te maken tijdens deze oefenzitting en jullie groepswerk.

Voor dat je aan de oefeningen begint, zal je via zelfstudie de basisbegrippen van Git moeten aanleren. We raden hiervoor aan tenminste de volgende secties te lezen uit het vrij beschikbare Pro Git boek:[11] 1.1, 1.5, 1.6, 2.1, 2.2, 2.3, 2.4 en 2.5. Als je Git thuis op je eigen computer wilt gebruiken, lees dan ook sectie 1.4. Beperk jezelf echter niet tot dit boek als je problemen hebt! Zoals boven al vermeld, is er online zeer veel informatie te vinden over Git.

2.2 Oefeningen

De eerste keer dat je een bepaald Git commando zal moeten gebruiken, zal dit commando tussen haakjes aangegeven worden. Welke parameters je aan het commando moet meegeven, zal je zelf moeten uitzoeken (`git help`). Als je later een keer hetzelfde commando nodig hebt, zal dit niet nogmaals worden aangegeven.

Simpele, lokale commando's

Begin met in je home directory een map met de naam `git` aan te maken waar we in kunnen experimenteren. Initialiseer in deze map een nieuwe Git repository met de naam `iwtest` (`git init`). Navigeer naar de nieuwe repository. Maak een nieuw bestand `tekst.txt` aan en vul dit met enkele lijnen tekst. Voeg dit bestand toe aan de repository (`git add`) en maak een eerste commit (`git commit`). Lees altijd de uitvoer van de commando's die je uitvoert! Als je bijvoorbeeld Git vergeten bent te configureren, zal het laatste commando je dit verteld hebben en zeggen wat je moet doen.

Door de commit zal het bestand `tekst.txt` nu veilig in de repository zitten. Om aan te tonen hoe zeker we ons hier over voelen, mag je nu het bestand verwijderen. Bekijk de inhoud van de repository om er zeker van te zijn dat het bestand verwijderd is. Bekijk de status van de repository om te weten wat je moet doen om het bestand te herstellen en voer dit uit (`git checkout`). Verifieer dat `tekst.txt` inderdaad juist hersteld is.

Voeg een aantal lijnen toe aan `tekst.txt` en commit de wijzigingen. Voeg een bestand `tekst2.txt` toe, gevuld met wat tekst, en maak weer een commit. Als je niet meer weet hoe je een bepaalde actie moet uitvoeren, kan het handig zijn de status van de repository te bekijken (`git status`), hier wordt vaak handige informatie vermeld. Als je niet meer weet hoe een commando werkt, kan je de documentatie lezen (`git help`). Bekijk nu de geschiedenis van de repository en verifieer dat er drie commits zijn gemaakt (`git log`).

Verwijder het bestand `tekst2.txt` weer en maak het definitief door te committen (`git rm`). Ook al is de verwijdering gecommit, je zal het bestand toch kunnen herstellen aangezien Git de volledige geschiedenis van elk bestand bijhoudt. Herstel het bestand en commit het. (Hint: gebruik `git log` om de *hash*, een uniek hexadecimaal nummer, van de laatste commit te vinden waar `tekst2.txt` nog bestond. Gebruik dan `git checkout` om het bestand te herstellen naar de toestand die het had bij die commit)

Werken met remote repositories

We kunnen het werken met een remote repository simuleren door lokaal een speciaal type repository aan te maken. Als we deze dan als remote toevoegen aan onze huidige repository, kunnen we alles doen als we met een "echte" remote ook kunnen doen. Je kan dit doen als volgt (als je dit niet begrijpt: maak je geen zorgen, dit zal je tijdens het groepswerk nooit moeten doen):

```
$ git init --bare ~/git/iwtest-remote
```

[11] `http://git-scm.com/book/`

```
$ git remote add origin ~/git/iwtest-remote
$ git config --global push.default simple
$ git push --set-upstream origin master
```

Na het uitvoeren van deze stappen is de repository in `~/git/iwtest` verbonden aan de remote repository in `~/git/iwtest-remote`.

We zullen nu gaan werken met de remote repository en tegelijkertijd een aantal problemen aanhalen die zich kunnen voordoen wanneer je met meerdere mensen met dezelfde remote werkt. Begin met een nieuwe repository aan te maken in `~/git/iwtest-clone` als clone van de `~/git/iwtest-remote` repository. (Hint: de URL die je aan het `git clone` commando geeft, kan ook het pad van een lokale map zijn.) Ga nu naar deze nieuwe repository, maak het bestand `tekst3.txt` aan, schrijf hier wat tekst in en commit het. Ga ook even terug naar de originele repository (in `~/git/iwtest`) en verifieer dat `tekst3.txt` daar nog niet bestaat. Onthoud dat een commit een lokale operatie is en dus nooit iets naar een remote repository stuurt! Zorg er nu voor dat de commit die je hebt gemaakt beschikbaar is in de remote (`git push`). Het nieuwe bestand is echter nog steeds niet beschikbaar in de originele repository, los dit op (`git pull`).

Voeg in de originele repository een lijn toe aan het bestand `tekst3.txt`, commit het en push de commit. Ga nu naar de clone van de repository maar *pull de verandering hier niet*! Voeg ook hier een lijn toe aan het bestand `tekst3.txt` (maar niet dezelfde als in de andere repository), commit het en push de commit. Git zal proberen om wijzigingen in verschillende delen van een bestand netjes te "mergen". Wijzigingen rond dezelfde regel in een tekst, zullen echter niet automatisch kunnen worden samengevoegd en Git zal je in dergelijke gevallen om hulp vragen. Lees de foutboodschap goed en los het probleem op. De oplossing zorgt voor het volgende probleem: er is een merge conflict. Dit kan ontstaan wanner twee of meer mensen hetzelfde bestand hebben gewijzigd en deze wijzigingen willen delen met een remote.

Een merge conflict moet je oplossen en er dan een commit van maken volgens de foutmelding. Het oplossen bestaat erin het conflicterende bestand aan te passen. Dit bestand ziet er nu ongeveer als volgt uit:

```
[originele inhoud]
<<<<<<< HEAD
[wijzigingen gemaakt in de huidige repository]
=======
[wijzingen die al gedeeld werden met de remote]
>>>>>>> 506d0b960a5f165f1f273196c73b25463eb8f6a4
```

Het komt er nu op neer dat je zal moeten kiezen hoe je wil dat het bestand er uit ziet: wil je beide wijzigingen houden of niet? De lijnen met `<<<<<<<`, `=======` en `>>>>>>>` zullen in ieder geval verwijderd moeten worden. Pas het bestand nu aan zodat beide wijzigingen behouden blijven en commit het. Nu zal het wel lukken om de wijzigingen de delen met de remote.

3 SSH

SSH (secure shell) laat je toe om in te loggen op een andere pc. Je kan dit nodig hebben om meerdere redenen, zoals bijvoorbeeld

- Enkel die andere pc heeft toegang tot de nodige bestanden. Dit kan het geval zijn als je thuis zit en je programma's wilt gebruiken die op de KUL-pc's staan, of aan gegevens wilt die op je KUL-opslagruimte staan.
- Je wilt meerdere langdurige berekeningen laten uitvoeren, die je dan verdeelt over verschillende pc's. Je kan jezelf natuurlijk fysiek van pc tot pc verplaatsen, maar je moet daarvoor ter plekke zijn en de pc zelf mag dan niet reeds in gebruik zijn door iemand anders.

Een lijst van de computers in de pc-klassen vind je op

`http://mysql.cs.kotnet.kuleuven.be`

Deze site is enkel bereikbaar van binnenuit het KUL-netwerk, als je dus thuis bent zal deze url vermoedelijk niet werken in je browser. Je kan gebruik maken van `lynx`, een browser die werkt in de terminal, of de html-pagina handmatig downloaden met `wget` en erop filteren:

```
wget mysql.cs.kotnet.kuleuven.be -O - | grep users
```

Log in op één van deze systemen:[12]

```
ssh [stadsnaam].cs.kotnet.kuleuven.be
```

Indien je een boodschap krijgt i.v.m. een RSA key fingerprint, kan je hierop "yes" antwoorden. Nadat je je paswoord hebt ingegeven, ben je ingelogd op de pc met de door jou gekozen naam.

Indien je van thuis uit inlogt, kan het zijn dat je via een tussenhalte moet werken: maak eerst een ssh-verbinding met `st.cs.kuleuven.be` en van daaruit kan je dan inloggen op de machines in de pc-klassen. Indien je met Windows werkt, kan je een SSH client downloaden vanop de ICTS-website:

- Ga naar `icts.kuleuven.be`.
- Onder "PC-dienstverlening en software" klik je op "Campuslicenties".
- Kies "X-Win32 server en F-Secure SSH Client for Windows".
- Volg de instructies.

Een andere SSH-client is PuTTY: `http://www.chiark.greenend.org.uk/~sgtatham/putty/`.

Wil je via de terminal bestanden kopiëren van de ene naar de andere pc via SSH, dan kan dit met `scp`:

```
scp [bestandsnaam] rXXXXXXX@[hostnaam]:/home/rXXXXXXX/
```

Bijvoorbeeld:

```
# Kopie lokaal bestand naar remote pc
scp result.txt rXXXXXXX@roeselare.cs.kotnet.kuleuven.be:/home/rXXXXXXX/

# Kopie remote bestand naar lokale pc
scp rXXXXXXX@roeselare.cs.kotnet.kuleuven.be:/home/rXXXXXXX/result.txt .
```

4 Verdere nuttige commando's

Unix biedt talloze commando's aan die je toelaten op een efficiënte manier met het besturingssysteem te werken. Wij kunnen jullie in deze oefenzitting slechts een kleine subset aanbieden. Toch wilden we jullie de volgende handige commando's niet onthouden.

4.1 Toegangsrechten

We gaan nakijken wat de toegangsrechten zijn tot je thuisdirectory. Hiervoor moet je naar de parent directory gaan van je thuisdirectory:

```
~ $ cd
~ $ cd ..
/home $
```

We zijn dus afgedaald van `/home/rXXXXXXX` naar `/home`. We vragen de inhoud aan:[13]

[12] Naargelang op welke machine je ingelogd bent, zal het al dan niet nodig zijn de extra `.cs.kotnet.kuleuven.be` toe te voegen.

[13] Het is mogelijk dat enkel jouw thuisdirectory opgenomen staat in de lijst. We besparen je de technische details, maar enkel de recent aangesproken thuisdirectories zijn zichtbaar. Na enkele minuten inactiviteit verdwijnen ze weer (zonder dat gegevens verloren gaan uiteraard). Binnen je eigen thuisdirectory zal dit niet gebeuren.

```
/home $ ls
[lijst van home directories]
```

Om te weten te komen welke toegangsrechten geassocieerd zijn met je thuisdirectory, moet je `ls` om extra informatie vragen:

```
/home $ ls -l
drwx------ 29 rXXXXXXX student 4096 2009-10-9 14:16 rXXXXXXX
```

We zijn geïnteresseerd in de linkerkolom. We maken de eigenlijke structuur duidelijker met spaties:

```
d rwx --- ---
```

De `d` betekent dat het gaat om een directory, maar dat wisten we al. Vervolgens komen de actuele toegangsrechten in drie groepjes van drie tekens. We onderscheiden drie niveaus gebruikers, waaraan we verschillende rechten kunnen toekennen:

- De gebruiker zelf: deze is de eigenaar van de directory, maar dit betekent niet dat hij zomaar alle rechten heeft. Hij kan zichzelf uit voorzorg bepaalde rechten ontnemen. Dit niveau komt overeen met het eerste groepje (`rwx` in ons geval).
- De groep waartoe de gebruiker behoort, in dit geval `student`. Dit is het tweede groepje (`---`).
- De rest van de gebruikers: het derde groepje (ook `---`).

user			group			other		
read	write	execute	read	write	execute	read	write	execute

We onderscheiden de volgende types toegangsrechten:

- `r`: leesrecht
- `w`: schrijfrecht
- `x`: uitvoerrecht

Dus, de rechten `drwx------` betekent dat enkel jij lees-, schrijf- en uitvoerrechten hebt. Stel dat je je medestudenten leestoegang wilt geven[14] tot je bestanden, dan kan je dit met `chmod`:

```
/home $ chmod g+r rXXXXXXX
/home $ ls -l
drwxr----- 29 rXXXXXXX student 4096 2009-10-9 14:16 rXXXXXXX
```

Lees `g+r` als "group: add read". Zo staat de eerste letter telkens voor het niveau (`u`, `g`, `o` of `a` voor alle drie niveaus tezamen), het tweede teken bepaalt of je rechten toevoegt (`+`) of verwijdert (`-`), en de overige letters over welke rechten je het hebt. Enkele andere voorbeelden:

```
  rwx --- ---
chmod u-wx rXXXXXXX      # Laat jezelf niet toe om te schrijven of uit te voeren
  r-- --- ---
chmod g+w rXXXXXXX       # Geef schrijfrechten aan andere studenten
  r-- -w- ---
chmod a+x rXXXXXXX       # Geef iedereen uitvoerrechten
  r-x -wx --x
```

Je kan de rechten terug naar hun oorspronkelijke toestand herstellen met[15]

```
/home $ chmod 700 rXXXXXXX
/home $ ls -l
drwx------ 29 rXXXXXXX student 4096 2009-10-9 14:16 sXXXXXXX
```

[14] Geen nood, later tonen we hoe je de toegangsrechten weer herstelt.

[15] `chmod` kan ook werken via bits: zo staat `rwx` gelijk met `111`, wat binair is voor `7`. Pas dit principe toe voor de drie niveaus, en je bekomt `700`.

4.2 Procesbeheer

In deze sectie bespreken we het beheer van processen. Als je een programma opstart, wordt door het besturingssysteem een nieuw proces aangemaakt, dat in feite de uitvoering van dit programma voorstelt. Met een proces gaan een aantal resources gepaard[16]:

- Een proces krijgt een eigen geheugenruimte.
- Binnen deze geheugenruimte wordt de code van het uit te voeren programma geladen.
- Er wordt een thread aangemaakt die de code gaat uitvoeren.
- De standaardstreams worden opgesteld (stdin, stdout, stderr).
- ...

Elk proces heeft ook zijn eigen id (een uniek identificatienummer), een huidige toestand (actief, gestopt, ...), en een prioriteit (bepaalt hoeveel processortijd deze toegekend krijgt.) Een snel overzicht van je eigen processen krijg je met `ps`:

```
~ $ ps
  PID TTY          TIME CMD
 5263 pts/0    00:00:00 bash
19984 pts/0    00:00:00 ps
```

Er zijn dus twee actieve processen op je naam: `bash` is de naam van de terminal, en `ps` is het programma dat je zojuist hebt opgeroepen. Je kunt ook een volledigere overzicht krijgen met het programma `top`, dat alle processen toont, de prioriteiten, het CPU- en RAM-verbruik, ... Je kan dit programma verlaten door op `q` te drukken.

Maak een nieuwe directory aan en maak er je working directory van (`mkdir` en `cd`). Voer het volgende commando uit:

```
wget http://people.cs.kuleuven.be/~job.noorman/oefenzittingen/iw/md5crack
```

Als je op een 32-bit system werkt:[17]

```
wget http://people.cs.kuleuven.be/~job.noorman/oefenzittingen/iw/md5crack_32 -O md5crack
```

Dit downloadt het bestand `md5crack`[18] dat te vinden is op de gegeven URL. Dit programma zal de MD5 hashfunctie op een brute-force wijze proberen te kraken. Dit wilt zeggen dat het programma gegeven een hashwaarde een string zal zoeken die, wanneer deze als input voor MD5 wordt gebruikt, deze hashwaarde oplevert.[19] Je kan het proberen uitvoeren met[20]

```
./md5crack 9dddd5ce1b1375bc497feeb871842d4b
```

maar dit zal in eerste instantie niet werken. Los dit probleem op. Welke input string wordt gevonden?

Vermits het programma vrij veel uitvoer produceert, kan je het best nogmaals starten maar deze keer de uitvoer laten wegschrijven naar een bestand. Open dit bestand in de terminal en probeer te ontdekken hoe het brute-force zoeken werkt.

[16] Voor meer informatie verwijzen we je naar het vak besturingssystemen.

[17] Dit kan je controleren via het `uname -a` commando.

[18] Verwar dit bestand niet met het bestand dat je eerder gedownload hebt voor een oefening. Dit bestand is immers extensieloos. Het is een gecompileerde versie van de code uit `md5crack.c`.

[19] Maak je geen zorgen, jullie hoeven voor deze oefenzitting niet te begrijpen wat een hashfunctie precies is. Onthoud enkel dat een goede hashfunctie onomkeerbaar is en dat brute-force zoeken dus in principe de enige manier is om de input te achterhalen. Geïnteresseerden kunnen meer informatie vinden op `http://en.wikipedia.org/wiki/Cryptographic_hash_function`.

[20] De `./` ervoor betekent "in de huidige directory". Deze expliciete vermelding van de directory is enkel nodig voor het uitvoeren van bestanden.

Aangezien de input string bij de vorige opdracht slecht drie letters lang was, ging het brute-force zoeken snel. Voor de volgende opdrachten willen we echter dat het zoeken wat langer duurt. Zoek daarom, via het `md5sum` commando, eerst de hashwaarde van de string `linux`. Laat het `md5crack` programma daarna proberen deze hashwaarde te kraken. Je ziet dat dit al wat langer duurt. Tijdens het rekenwerk blokkeert de terminal ook: je kan in de tussentijd geen andere opdrachten meer laten uitvoeren. Je kan het programma gewoon onderbreken met `CTRL-C`, maar je kan het ook pauzeren met `CTRL-Z`. Je zal dan met `ps` kunnen zien dat `md5crack` in de lijst van processen staat.

Een concept dat gelijkaardig is met processen is de job. We besparen je de details, maar het komt erop neer dat je met het `jobs` commando een lijst kan zien van alle actieve jobs. Het is dus vergelijkbaar met `ps`, alleen dat jobs andere getallen toegekend krijgen. We moeten dus een verschil maken tussen de job id en de process id:

```
$ ps
  PID TTY          TIME CMD
 5263 pts/0    00:00:00 bash
13923 pts/0    00:00:00 md5crack
14355 pts/0    00:00:00 ps

$ jobs
[1]+  Stopped         ./md5crack e206a54e97690cce50cc872dd70ee896
```

Je kunt de `md5crack` job weer actief maken met het volgende commando:

```
fg 1
```

maar dit zorgt er weer voor dat je terminal geblokkeerd raakt tot het programma eindigt of onderbroken wordt met `CTRL-C` of `CTRL-Z`. Daarom kunnen we het programma ook laten draaien op de achtergrond:

```
bg 1
```

Zo kan je tijdens de uitvoering van het programma toch nog gebruik maken van de terminal. Vervelend is wel dat het programma zijn uitvoer naar de terminal blijft wegschrijven, wat nogal verwarrend kan overkomen. Daarom is het sowieso beter ervoor te zorgen dat je de stdout omleidt, bv. naar een bestand met de `>` operator.

Je kan een job ook definitief onderbreken met het `kill` commando:

```
kill %1
```

`kill` neemt normaalgezien een process id als argument, maar door middel van het `%`-teken kunnen we er ook naar verwijzen d.m.v. de job-id. We hadden dus even goed kunnen schrijven

```
kill 13923
```

waarbij we de `13923` eerder verkregen via `ps`. Het is mogelijk (en vooral praktischer) om een job rechtstreeks op de achtergrond te laten uitvoeren i.p.v. het op te starten, te pauzeren met `CTRL-Z` en er dan `bg` op toepassen. Dit kan door gebruik te maken van de `&` operator:

```
./md5crack e206a54e97690cce50cc872dd70ee896 > result.txt &
```

Stel nu dat je de hashwaarde van een nog langer input string wil kraken. Dit kan gemakkelijk uren of zelfs dagen in beslag gaan nemen. Er duiken drie problemen op:

- Je wil niet blijven wachten tot de berekening voorbij is, maar je wil ook niet ingelogd blijven (waardoor iemand anders aan je account kan). Uitloggen kan echter de berekening stopzetten. Je moet dus een manier vinden om te vermijden dat de job onderbroken wordt wanneer je uitlogt.

- Andere gebruikers willen nog gebruik kunnen maken van de pc waarop je programma draait. Hoewel het programma op de achtergrond draait, zal het systeem zal toch voelbaar trager zijn. Een admin kan ervoor kiezen je programma te killen[21].

- Je wil graag automatisch een boodschap krijgen als de berekening voorbij is i.p.v. telkens te moeten gaan checken.

Het eerste probleem verhelp je met `nohup`. Bekijk de man pages om te weten te komen hoe dit commando werkt. Voor het tweede probleem gebruik je `nice` of `renice`, zoek hiervan tevens de werking op. Voor het laatste probleem kan je het resultaat van de berekening automatisch laten mailen naar jezelf met `mail`. Probeer dit eens te doen.

In het geval dat de geproduceerde uitvoer te groot is om te mailen, kun je het enerzijds eerst comprimeren (zoek hiervoor `tar` of `zip` op) en versturen als attachment (gebruik `mutt` met de `-a` optie), of anderzijds kan je het resultaat gewoon laten wegschrijven naar een lokaal bestand en een korte boodschap via mail sturen (`echo 'Berekening voltooid' | mail [adres]`). Je zal de sequentieoperator `;` nodig hebben, die toelaat opdrachten na elkaar uit te voeren zonder streamomleidingen. Een bash-script aanmaken is ook een optie (zoek online naar meer informatie).

5 Linux thuis proberen

Er zijn meerdere manieren om Linux op je eigen pc te gebruiken.

- Je kan meteen gaan voor een volledige Linux-installatie: dit houdt in dat je Linux op je harde schijf installeert naast Windows (je kan Windows natuurlijk ook gewoon overschrijven), en via een boot-manager kan je dan kiezen bij het opstarten van de pc welk besturingssysteem je wilt opstarten.

- Een minder ingrijpende manier is gebruik te maken van een live CD: Linux draait vanop de CD en laat de harde schijf onaangeroerd. Er kan dus minder mislopen, maar het is geen lange termijnoplossing omdat er een aantal nadelen aan zijn gebonden: het booten gebeurt traag, je kan het besturingssysteem niet zomaar upgraden of programma's installeren, ... Live CDs zijn eerder bedoeld om het besturingssysteem snel uit te testen, en op basis van deze ervaring een meer definitieve keuze te maken.

- Je kan Linux laten draaien op een virtuele machine. We bespreken deze methode uitvoeriger in de volgende sectie.

De twee eerste aanpakken zijn relatief riskant (zeker vergeleken met de derde): je moet voor beide vermoedelijk het BIOS ingaan en de boot sequence aanpassen, maar als je dan per ongeluk andere instellingen wijzigt kan dit je pc onklaar maken. Bij het installeren van Linux op de harde schijf zal je mogelijk moeten herpatitioneren, en als je hier niet mee oppast kan je al je gegevens kwijt zijn, etc. We wensen daarom te benadrukken dat voor welke aanpak je ook kiest, je dit op eigen risico doet, en we niet verantwoordelijk zijn voor eventuele schade.

5.1 Werken met een virtuele machine

Je kan virtualisatiesoftware gebruiken om virtuele machines die een pc softwarematig simuleren aan te maken. Je kan instellen hoeveel RAM, hoeveel CPU cores, ... een virtuele machine toegewezen krijgt. Je maakt vervolgens een virtuele harde schijf aan, die op je eigen pc een gewoon bestand is, maar voor de VM een harde schijf voorstelt. Nadien kun je op deze VM een besturingssysteem naar keuze installeren.

Er zijn tal van voordelen verbonden met deze aanpak:

- Het is veilig voor leken omdat de VM volledig "opgesloten" wordt in zijn eigen wereldje, en kan niet zomaar aan de rest van je pc (tenzij je daarvoor expliciet toelating geeft.) Als er dus iets ernstig

[21] En dit gebeurt wel degelijk.

misloopt binnen de VM, is het gewoon een kwestie van de VM te verwijderen en er een nieuwe aan te maken.

- Vermits de virtuele harde schijf een gewoon bestand is, kan je dit gemakkelijk kopiëren als backup van het ganse guest besturingssysteem (= het besturingssysteem dat je installeert in de VM). Dit laat je toe om veilig te experimenteren met allerlei instellingen. Met sommige VM-programma's is het mogelijk om met "differencing images" te werken: je splitst de virtuele harde schijf dan op in een onveranderlijke *parent image* en een wijzigbare *child image*. Alle wijzigingen die je dan doorvoert zullen beperkt worden tot de child image. Op deze manier kan je gemakkelijk meerdere kopies hebben van het besturingssysteem zonder er veel harde schijfruimte voor te moeten opgeven.
- Bij het echt installeren van een besturingssysteem op je fysieke harde schijf zit je altijd met het dilemma "hoeveel GB ken ik eraan toe?". Deze keuze maak je best op voorhand, want herpartitioneren is een niet-triviale operatie. Bij virtualisatie bestaat dit probleem niet: je kan gerust een virtuele harde schijf aanmaken van 100GB, op je fysieke harde schijf neemt deze slechts een fractie hiervan in beslag (voor zover de virtuele harde schijf weinig gegevens bevat, uiteraard).
- Je kan VMs kopiëren van de ene naar de andere pc.
- Je kan snel wisselen tussen 2 (of meer) besturingssystemen, zonder telkens te moeten rebooten. (Geloof ons, dit is een zeer groot voordeel.)
- Doorgaans zijn er "guest additions" die speciale drivers installeren in het guest besturingssysteem, waardoor er betere integratie is tussen guest en host besturingssysteem: zo kan eenzelfde muiscursor gebruikt worden voor beide besturingssystemen, kan het klembord gedeeld worden, kan er rechtstreeks toegang verleend worden tot de harde schijf, ...

Uiteraard zijn er ook wat nadelen aan verbonden:

- Beide het host en guest besturingssysteem moeten tegelijkertijd in het geheugen passen. Je hebt dus relatief veel RAM nodig.
- Het guest besturingssysteem dat draait in de VM zal ook wat trager werken dan normaal.
- De virtuele hardware is meestal beperkt. Het is goed mogelijk dat de guest besturingssysteem niet optimaal gebruik zal kunnen maken van je hardware (bv. geluidskaart, 3D-kaart, scanner, ...)

5.2 Software

Ziehier een korte lijst van (gratis) virtualisatiesoftware:

- VirtualBox: `http://www.virtualbox.org/`
- VirtualPC: `http://www.microsoft.com/windows/virtual-pc/support/virtual-pc-2007.aspx`
- VMWare Player: `http://www.vmware.com/products/player/`

Je zal ook een CD image (een ISO bestand) moeten downloaden die zal fungeren als "virtuele CD" in de VM. Een populaire Linux-distributie is Ubuntu, waarvan je de ISO kan vinden op `http://www.ubuntu.com/getubuntu/download`.

Standaard kies je best telkens voor de 32 bit versie. Wil je een 64 bit versie gebruiken, moet je huidig besturingssysteem zelf ook 64 bit zijn, moet je de 64 bit versie van de virtualisatiesoftware gebruiken (als die tenminste bestaat), en moet je de 64 bit CD image downloaden. Hou er rekening mee dat als je programma's compileert onder 64 bit Linux, deze niet zomaar zullen werken op 32 bit versies van het besturingssysteem (maar wel vice versa). De KUL zelf draait doorgaans 32 bit versies van Linux (en Windows), dus als je je programma's wilt laten draaien op de pc's van de KUL, kan je best kiezen voor een virtuele 32 bit Linux.