Programmeertechnieken
Week 1

Tim Cocx, Kristian Rietveld

http://liacs.leidenuniv.nl/~rietveldkfd/courses/pt2016/

Universiteit Leiden
The Netherlands

Even voorstellen …

- Docenten:
  - Tim Cocx
  - Kristian Rietveld
- Assistenten:
  - Tim van der Meij
  - Dennis van der Zwaan
  - Ruben Meerkerk

Website

- Alle informatie is terug te vinden op de website.
- Hier is ook een “ICS” link te vinden met alle deadlines.

http://liacs.leidenuniv.nl/~rietveldkfd/courses/pt2016/

Programmeertechnieken

- Een nieuw vak in de bachelor Informatica.
- Vervolg op Programmeermethoden.
- Doel: programmeervaardigheden verder ontwikkelen.
- Programmeren is een gereedschap dat je in je verdere studie en professionele loopbaan zult gebruiken.

Leerdoelen

- UNIX tools & pipelines.
- Building & linking. Het kunnen gebruiken van bestaande bibliotheken.
- Pointeraritmatiek & bitwise manipulations.
- Advanced & Modern C++.
- Concepten object-georienteerd programmeren, design patterns.
- Software testen.
- Java, gebruik maken van bestaande Java frameworks.

Examinering

- Er is geen tentamen.
- Alle toetsing vindt plaats via opdrachten.
- Gevolg: de opdrachten zijn uitgebreid en vergen veel tijd! De meeste werkdruk voor dit vak zit dus gedurende het semester!
- Werkcolleges zijn een moment om vragen te stellen. Je hebt naast de werkcolleges een *groot* aantal uren nodig om de opdrachten te voltooien.

Opzet

- Het vak bestaat uit:
  - Een hoorcollegereeks.
  - Viertal practicumopdrachten.
  - Drietal huiswerkopgaven.
  - Werkcolleges, vaak in het teken van de opdrachten, soms een speciaal thema.

Overzicht opdrachten & deadlines

- Viertal practicumopdrachten:
  - Data Analysis Pipelines. [15%, **19 februari**]
  - "Spreadsheet" (C++). [30%, **8 april**]
  - "Vang de volger" (Java, Swing). [25%, **29 april**]
  - "Volg je vrienden" (Java, Play, Android). [30%, **27 mei**]
- Een interview maakt deel uit van de beoordeling van de laatste opdracht.
- Alle opdrachten moeten afzonderlijk voldoende zijn (>= 5.5).

Overzicht huiswerk & deadlines

- Drietal huiswerkopgaven:
  - "Shared Library maken" [**12 februari**]
  - "Pointeraritmatiek & bitwise operators" [**26 februari**]
  - "Unit testing" [**1 april**]
- Maximaal 4 uur werk.
- Huiswerk telt niet mee voor het eindcijfer, maar moet voldoende zijn om het vak te kunnen halen.

Fraude & plagiaat

- Zoals gezegd bij programmeermethoden: assistenten om hulp vragen mag, mede-studenten vragen stellen mag, code overnemen is ***zeer streng verboden***.
- Code voor jezelf houden en niet delen met anderen, ook niet in de komende jaren.
- Wissel nooit code met elkaar uit, ook niet om te helpen! Help elkaar door vragen te stellen en zaken uit te leggen.
- Alle inzendingen zullen automatisch worden gecontroleerd op plagiaat.

Advanced UNIX shell usage

Programma's en processen

- Computers worden gebruikt door er programma's op uit te voeren.
- Programma bestaat uit:
  - Een reeks van instructies.
  - Initialisatiedata.
- Een proces bestaat uit een programma *en* toestand ('state').
- Het opstarten van programma's wordt gedaan door een besturingssysteem.

Shells

- Om te interacteren met een besturingssysteem, maken we gebruik van een programma dat een "shell" (schil) wordt genoemd.
- Met behulp van de shell kunnen we programma's opstarten.
- Er bestaan zowel grafische als tekst-gebaseerde shells.
- In dit college zullen we ons beperken tot de tekst-gebaseerde shells. Deze zijn programmeerbaar en hebben vele functionaliteiten.

Tekst-gebaseerde shells

- In het Engels vaak: "command-line interface" (CLI).
- De user interface bestaat uit een invoerprompt waarin commando's kunnen worden ingevoerd.
- Over het algemeen zijn deze commando's namen van programma's die moeten worden opgestart.

Standaardprogramma's

- Op UNIX systemen zijn er een groot aantal standaard programma's te vinden.
- Deze programma's zijn er voor allerlei taken:
  - Lijst van bestanden weergeven (ls).
  - Bestanden verplaatsen (mv).
  - Kopieren (cp).
  - Lijst van processen opvragen (ps).
  - De computer uitzetten (poweroff).

Verschillende shells

- Net als er verschillende besturingssystemen bestaan, zijn er ook meerdere tekst-gebaseerde shells geschreven.
- Populair zijn:
  - Bash
  - tcsh
  - zsh
  - tcsh
  - (Windows: PowerShell)
- De meeste Linux machines starten standaard "bash" op.

Invoerprompt

- Er zijn allerlei handige trucs die je kunt gebruiken bij het invoerprompt:
  - Pijlen links en rechts.
  - Ctrl-a: ga naar begin regel, Ctrl-e: ga naar einde regel.
  - Ctrl-k op begin regel: alles weghalen ("kill", eigenlijk "knippen").
  - Ctrl-y: "yank" (plakken).
  - Pijlen omhoog en omlaag: door geschiedenis bladeren.
  - Ctrl-r en intypen zoekterm: door geschiedenis zoeken (!).

Tab completion

- Door op "Tab" te drukken zal de shell proberen het huidige woord automatisch aan te vullen.
- Handig als je iets niet meer uit je hoofd weet.

Invoer & uitvoer

- Veel programma's produceren uitvoer.
- Vaak kan er worden gekozen of dit of naar een bestand wordt geschreven, of naar *stdout* of *stderr*.
- Zelfde voor invoer: keuze of dit vanuit een bestand wordt gelezen of *stdin*.

Pipes

- Je kunt ook de *stdout* van het ene programma knopen aan de *stdin* van een ander programma.
- Dit wordt gewoon met het symbool "|" (pipe).
- Voorbeeld:

```
ls /usr | sort | less
```

- Een dergelijke aaneenschakeling wordt ook wel een "pipeline" genoemd.
- "less" vangt uitvoer op zodat je er rustig doorheen kunt scrollen.

Redirectie

- Binnen de shell kun je ervoor zorgen waar de uitvoer een programma "naar toe" gaat.
- Als een programma de uitvoer naar *stdout* schrijft, kan de shell ervoor zorgen dat dit wordt "omgeleid" naar bijvoorbeeld een bestand.
- In het volgende voorbeeld wordt de uitvoer van *ls* naar *stdout* omgeleid naar een bestand *uitvoer.txt*.

```
ls -al /usr > uitvoer.txt
```

Redirectie (2)

- Met ">" wordt de *stdout* van een proces naar een bestand gestuurd.
- Met "<" wordt de inhoud van een bestaand bestand naar de *stdin* van een proces gestuurd.
- Merk op dat pipes en redirection mogen worden gecombineerd.

```
cat bestand | sort > gesorteerd
cat bestand | sort | uniq > unieken
sort < bestand
sort < bestand > gesorteerd
```

Redirectie (3)

- Standaard zal ">" een bestaand bestand geheel vervangen.
- Om dit te voorkomen: ">>" voor output en append.

```
cat bestand1 > alles
cat bestand2 >> alles
cat bestand3 >> alles
```

Redirectie (4)

- We kunnen nog een stap verder gaan en ook specifieke *file descriptors* selecteren voor invoer of uitvoer.
- In het algemeen geldt:
  - 0 = stdin
  - 1 = stdout
  - 2 = stderr

```
gcc fout.c > warnings   # werkt niet!
gcc fout.c 2> warnings
```

Redirectie (5)

- Je kunt file descriptors ook dupliceren.
- Syntax:
  - "2>&1" file descriptor 2 is nu een kopie van 1.
- Alles wat wordt geschreven naar *stderr*, komt nu terecht in *stdout*.
- Volgorde is belangrijk!

```
gcc fout.c | less   # werkt niet!
gcc fout.c 2>&1 | less
gcc fout.c > warnings 2>&1
```

Pipelines bouwen

- Piping en redirection gecombineerd met de vele standaardcommando's die op UNIX systemen beschikbaar zijn, vormen een zeer handigere gereedschapkist!
- Allerlei zaken kun je op deze manier heel vlot automatiseren.
- In de eerste opdracht gaan we oefenen met het bouwen van "pipelines".

Handige commando's

- cat: Bestanden lezen
- zcat: Gecomprimeerde bestanden lezen
- sort: Sorteren
- uniq: Van elk maar 1 doorlaten (duplicaten verwijderen)
- less: Pager
- grep: Filteren
- sed: "Stream editor"
- cut: Stukken uit regels verwijderen
- wget: Downloaden van URLs.

Handige commando's (2)

```
cat bestand | grep " is " | less
cat bestand | grep "^a" | less
cat bestand | sed "s/is/was/" | less
cat bestand | cut -b 1-4,6
wget http://www.test.nl/test.txt
```

Handige commando's (3)

- Er zijn uiteraard veel meer van dit soort utilities.
- Zoeken kan met "*man -k <woord>*".
- Vervolgens kun je de handleiding opvragen met "*man <commando>*".
- Uiteraard kun je ook zelf dit soort utilities schrijven.
  - Vaak in een "scripttaal": zie volgende week.

Inpakken en uitpakken

- Op UNIX systemen standaard manier om directories in- en uit te pakken.
- We maken gebruik van "tar" (Tape ARchive).
- Een gehele directory structuur kan worden gebundeld in 1 bestand.
- Daarna comprimeren met gzip (z), bzip2 (j) of xz (J).

Inpakken en uitpakken (2)

- Voorbeeld inpakken:

```
tar czvf ingepakt.tar.gz <directory>
```

- Verklaring der tekens:
  - "c": create, "z": zip, "v": verbose, "f": filename.
- Het resultaat, `ingepakt.tar.gz`, noemen we een "tarball".

Inpakken en uitpakken (3)

- Voorbeeld uitpakken:

```
tar xzvf ingepakt.tar.gz
```

- Inhoud bekijken zonder uitpakken:

```
tar tzvf ingepakt.tar.gz
```

Uitpakken (4)

- Soms worden niet-tar bestanden ingepakt met gzip of bzip2. Deze zijn te herkennen aan de extensie ".gz" of ".bz2".
- De volgende tools zijn handig om met dit soort gecomprimeerde bestanden te werken:
  - gz: zcat, zless, zgrep
  - bz2: bzcat, bzless, bzgrep

Shells op "remote" hosts

- Toe nu toe hebben we gewerkt met een shell op de lokale machine.
- We kunnen ook een shell "openen" op een andere machine, een "remote" host.
- Er wordt via het Internet een verbinding gemaakt waarover de invoer naar de andere computer wordt gestuurd en de uitvoer wordt naar ons gestuurd.

Shells op "remote" hosts (2)

- We doen dit met "ssh": "secure shell".
  - (Het is de opvolger van "rsh", "remote shell").

```
ssh username@hostname
```

- Windows gebruikers: download Putty!

Kopieren naar andere machines

- We kunnen ook bestanden kopieren naar een andere computer.
- "scp": "secure copy".

```
scp bestand.txt user@hostname:/home/user/bestand.txt
```

- "-r" werkt voor recursief kopieren.
- Met "sftp" krijg je een prompt waarmee je rond kunt kijken op de remote machine.
- Windows gebruikers: bekijk eens "winscp".

ssh Universiteit Leiden

- Er is een standaard "ssh gateway":

```
ssh sXXXXXXX@sshgw.leidenuniv.nl
```

- En vanaf daar moet je inloggen op een andere machine om werk te kunnen doen. Bijvoorbeeld de "huisuil":

```
ssh remotelx.liacs.leidenuniv.nl
```

Wildcards

- Om in de shell meerdere bestanden te "selecteren" kun je gebruik maken van *wildcards* (of "globbing patterns").
  - "*": shell mag 0 of meer karakters zelf invullen.
  - "?": shell mag precies 1 karakter zelf invullen.

```
file*.txt
    file1.txt, file.txt, file2.txt, fileA.txt,
  file1324.txt, filesdfasdf.txt, etc.
```

```
file?.txt
    file1.txt, file2txt, fileA.txt
```

Wildcards (2)

- OPPASSEN!

```
mv file*.txt
```

- Wat gebeurt hier?
- In het geval van onzekerheid: gebruik de interactive mode ("-i").

Gebruik variabelen

- Je kunt ook variabelen aanmaken en gebruiken:

```
a=1
blabla="sdf"
echo $a $blabla
```

- LET OP: geen spaties rond het "=" teken.

Environment variabelen

- Sommige variabelen beinvloeden het functioneren van het systeem.
- Dit zijn “environment variabelen” en deze zijn “geexporteerd”.
- Bijvoorbeeld:
  - "USER": bevat de naam van de huidige gebruiker.
  - "HOME": bevat de home directory van de huidige gebruiker.
  - "PATH": bevat het zoekpad dat de shell gebruikt bij het zoeken naar executables.
  - "EDITOR": bevat de naam van de editor die de gebruiker het liefst gebruikt.
  - "PS1": formaatstring van het invoerprompt.
- Bekijken en exporteren: commando `export`

Verschil in quotes

- Er worden drie verschillende quotes gebruikt in bash met ieder een karakteristieke werking.
  - "..": interpreteer de variabelen
  - '..': geen interpretatie van de variabelen.
  - `..`: interpreteer variabelen, interpreteer vervolgens als commando en voer dit commando uit. De originele string wordt vervangen met de uitvoer van het commando.

```
a=1
echo "ls $a"
echo 'ls $a'
echo `ls $a`
a=/usr
echo `ls $a`
```

Control flow

- In bash kunnen simpele programma's worden geschreven. Er zijn dus "compound commands" voor control flow.
- Je kunt bijvoorbeeld direct achter het invoerprompt een for loop schrijven.

Control flow (2)

- Een for loop itereert over een lijst elementen.

```
for i in 1 2 3; do
      echo $i;
done

for i in ls -1; do
      echo $i;
done

# Gebruik van backticks: `ls -1` zal worden     #
vervangen met de uitvoer van het uitvoeren
# van "ls -1".
for i in `ls -1`; do
      echo $i;
done
```

Control flow (3)

- "if" kiest een lijst van commando's om uit te voeren gebaseerd op de exit status (return code) van de conditie.
- Bij de conditie wordt vaak gebruik gemaakt van het programma "test" of "[".

```
if [ x$bla == "x4" ]; then
    echo "vier"
else
    echo "niet vier"
fi

if [ -e /etc/redhat-release ]; then
    echo "RedHat release file exists"
fi
```

Shell scripting

- Pipelines of for-commando's kunnen lang worden en je wilt deze niet keer op keer intypen.
- We kunnen deze hergebruiken door een script file te maken.
- Deze scripts kunnen dan worden uitgevoerd alsof het programma's zijn, hiervoor moet het bestand van "executable" zijn:

```
chmod +x <script file>
```

Shell scripting (2)

- De "shebang" regel ("#!") aan het begin van het bestand geeft aan welk programma het script kan uitvoeren.
- *$1, $2* enzvoort bevatten de argumenten aan het script.
- *$** bevat alle argumenten
- *$?* bevat de exit code van het laatst uitgevoerde programma.

```
#!/bin/bash
echo $1 $2
echo $*
cat $1 | sort | uniq > $2
```

awk

- Awk is een kleine programmeertaal voor het verwerken van (gestructureerde) tekst.
- Elke regel wordt beschouwd als een record en wordt opgesplitst in velden.
- Er kan worden gekozen op welk karakter wordt gesplitst.
- Vervolgens kun je acties definiëren die moeten worden uitgevoerd als er aan een bepaalde conditie wordt voldaan.
- De conditie kan ook worden weggelaten.

awk (2)

- Een regel "awk" ziet er als volgt uit:

```
condition { action }
```

awk (3)

- Twee eerste voorbeelden.
  - Print alle regels, zonder conditie.

```
awk '{ print }' bestand.txt
```

  - Print de regel alleen als deze begint met w:

```
awk '/^w/ { print }' bestand.txt
# zelfde als:  grep "^w" bestand.txt
```

awk (4)

- Uiteraard kan het programma uit meerdere condities en acties bestaan:

```
awk '/^W/ { print "W regel" } /^G/
{ print "G regel" }' bestand.txt
```

awk (5)

- Gedetecteerde velden kunnen worden benaderd via variabelen waarvan de naam begint met een dollar-teken.
  - *$0*: de hele regel.
  - *$1*, *$2*, .., *$n*: veldnummer "n".
  - Gebruik in bash de goede quotes! Gebruik anders het escape-karaketer: "\$".

```
echo "een twee drie" | awk '{print $2}'
echo "een twee drie" | awk "{print \$2}"
```

awk (6)

- Er zijn in awk ook een aantal speciale variabelen gedefinieerd:
  - *NR*: record number.
  - *NF*: aantal velden in een input record.
  - *FNR*: record number in huidige file.
  - *FS*: field separator die in gebruik is (regular expression).
  - *RS*: record separator die in gebruik is (regular expression, standaard '\n').

awk (7)

- Ook zijn er twee speciale condities die kunnen worden gebruikt:
  - Met *BEGIN* kan je een actie maken die aan het begin van het programma wordt uitgevoerd.
  - Daarnaast is er ook een conditie *END*.

```
echo "een;twee;drie" | awk 'BEGIN {FS=";"} {print $2}'
echo "een;twee;drie" | awk '{print $2}'
```

awk (8)

- Voorbeeld: tel het aantal regels dat met W of G begint.

```
/^W/ { w++; }
/^G/ { g++; }

END {
      print "G regels:", g;
      print "W regels:", w;
}
```

awk (9)

- awk beschikt ook over een array type. Dit zijn eigenlijk "dictionaries".
- Je kunt zelf kiezen wat je als subscript gebruikt, dit mag een string zijn, integer of iets anders.

```
telefoon["Holm"] = "06-12345678"
telefoon["Kris"] = "06-87654321"
print telefoon["Holm"]
```

awk (10)

- Awk kan nog veel meer: er zijn ook *if*, *for*, *while* statements. En een variant van *printf*.
- Een uitgebreid voorbeeld:

```
BEGIN {
    print "ARGC =", ARGC
    for (k = 0; k < ARGC; k++)
        print "ARGV[" k "] = [" ARGV[k] "]"
}

{ telefoon[$1] = $2 }

END {
    for (k in telefoon)
        print k ":", telefoon[k]
}
```

Na de pauze, practicum in zalen 302/304 en 306/308.