

Programmeertechnieken Week 1

Tim Cocx, Kristian Rietveld

<http://liacs.leidenuniv.nl/~rietveldkfd/courses/pt2017/>



Universiteit Leiden
The Netherlands

Even voorstellen ...

➤ Docenten:

- Tim Cocx
- Kristian Rietveld

➤ Assistenten:

- Leon Helwerda (AIO)
- Lars Suanet
- Vincent van Rijn
- Vincent den Hamer
- Elgar van der Zande
- Ruben Turkenburg (vragenuren)

Website

- Alle informatie is terug te vinden op de website:

<http://liacs.leidenuniv.nl/~rietveldkfd/courses/pt2017/>

Programmeertechnieken

- Nieuw vak, sinds vorig jaar gegeven.
- 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-georiënteerd programmeren, design patterns.
- Software testen.
- Java, gebruik maken van bestaande Java frameworks.

Opzet

Het vak bestaat uit:

- Een hoorcollegereeks.
 - Donderdagen, 5e en 6e uur Snellius B02; indien te druk in het van Steenis gebouw.
- Werkcolleges, vaak in het teken van de opdrachten, soms een speciaal thema.
 - Ondersteuning tijdens werkcolleges: donderdagen 7e en 8e uur computerzalen Snellius gebouw.
- Viertal practicumopdrachten.
- Drietal huiswerkopgaven.

Examinering

- Er is **geen** tentamen.
- Alle toetsing vindt plaats middels opdrachten.
 - Alle opdrachten moeten afzonderlijk voldoende zijn (≥ 5.5).
 - In geval van onvoldoendes: **twee** opdrachten mogen worden herkanst.
 - De gegeven deadlines zijn strikt: te laat ingeleverd werk wordt **niet** beoordeeld. Opnieuw/later inleveren als herkansing.

Overzicht opdrachten & deadlines

- Viertel practicumopdrachten:
 - Data Analysis Pipelines. [15%, **24 februari**]
 - "Spreadsheet" (C++). [30%, **deel A: 24 maart, deel B: 14 april**]
 - "Vang de volger" (Java, Swing). [25%, **5 mei**]
 - "Volg je vrienden" (Java, Play, Android). [30%, **2 juni**]
- Een interview maakt deel uit van de beoordeling van de laatste opdracht.
- Opdracht 1: groepen van 2, Opdracht 2 en verder: groepen van 3.

Overzicht huiswerk & deadlines

- Drietal huiswerkopgaven:
 - "Shared Library maken" [**17 februari**]
 - "Pointeraritmatiek & bitwise operators" [**3 maart**]
 - "Unit testing" [**31 maart**]
- Maximaal 4 uur werk.
- Mag worden gemaakt in duo's.
- Huiswerk telt niet mee voor het eindcijfer, maar moet voldoende zijn om het vak te kunnen halen.
- Huiswerk is bedoeld voor thuis en niet voor tijdens de werkcolleges.

Tijdplanning

- Omdat het vak **geen** tentamen heeft, zijn de programmeeropdrachten uitgebreid en vergen veel tijd!
- Alle werkdruk voor dit vak zit dus **gedurende** het semester!
 - Er is dus niet echt een “eindsprint voor het tentamen”.
- Werkcolleges zijn een moment om vragen te stellen. Je hebt naast de werkcolleges een **groot** aantal uren nodig om de opdrachten te voltooien.
 - Ga ervan uit dat je buiten de contacturen om iedere week 5 a 6 uur aan de opdrachten dient te besteden.

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.

Roosterwijzigingen

- Vaststellen tijdstip in de week voor een vragenuur.
 - Wat werkt? Dinsdagmiddag? Woensdagmiddag?
- Er zijn extra werkcolleges ingeroosterd:
 - Woensdag 26 april, 7e en 8e uur.
 - Woensdag 24 mei, 7e en 8e uur.
 - Donderdag 1 juni, 7e en 8e uur.
- Het weekschema op de website van het vak zal up-to-date worden gehouden met de laatste wijzigingen.

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
 - (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 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 op je gemak 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 > unieke  
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 handige 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 (.gz) lezen
 - Voor .bz2 gebruik je “bzip2”.
- 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.
- wc: word count, “wc -l” doet een regeltelling

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 (optie "z"), bzip2 (optie "j") of xz (optie "J").

Inpakken en uitpakken (2)

- Voorbeeld inpakken:

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

- Verklaring der opties:
 - "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: bzip2, 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 kopiëren naar een andere computer.
- "scp": "secure copy".

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

- "-r" werkt voor recursief kopiëren.
- 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 beïnvloeden 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`

Vershil 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 `seq 1 3`; do # seq maakt een getallenreeks
    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`, `$2` enzovoort bevatten de argumenten aan het script.
- `$*` 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 voor een regel 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 }
```

- De conditie wordt getest voor iedere regel.

awk (3)

Twee eerste voorbeelden:

- Print alle regels, zonder conditie.

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

- Print de regel alleen als deze begint met een “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 juiste quotes! Gebruik anders het escape-karakter: `"\$"`.

```
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 -F ';' '{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;  
}
```

- Een awk-programma mag in een apart bestand worden gezet, bijv. “prog.awk”.
- Om uit te voeren: `awk -f prog.awk bestand.txt`

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]  
}
```