Datorkunskap för IT-studenter

Övningshäfte

[1] Denna textrad är det enda som står på denna sida och det är avsiktligt.

Bakgrund och förord

Här är häftet med övningar till kompendiet Datorkunskap för IT-studenter.

Dator

1. Nämn olika typer av datorer som du använder
    a. Vad skiljer dem åt?
    b. Hur skiljer sig ditt användande av dem åt?
    c. Hur interagerar du med dem?
    d. Har de externa hårdvaror eller sitter allt ihop?
    e. Har de internetuppkoppling?
    f. Vilka är de vanligaste programmen du använder på dem?
    g. Hur mycket internminne är det i dem?
    h. Hur mycket hårddiskutrymme är det på dem? (Kallas ibland lagringsutrymme eller på engelska “storage”)
    i. Vilken sorts CPU är det i dem?
    j. Vilket operativsystem är det i dem?
        i. Vilken version av operativsystemet?
        ii. Vem tillverkar operativsystemet?

Programmering

1. Kan du räkna upp några programmeringsspråk?
    a. Kan du ta reda på huruvida de är kompilerade, interpreterade eller bytecode-kompilerade?
2. Kan en dator direkt (utan hjälp av annat program) köra källkoden som är skriven i ett programmeringsspråk?
    a. Om inte, varför?
    b. Om ja, hur?
3. Nämn några arbetsroller eller yrken som programmerare kan ha.
4. Vem brukar kallas “den första programmeraren”?
5. Nämn några vanliga sysslor programmerare ägnar sig åt, förutom just att skriva källkod.
6. En dator har en komponent som heter CPU (kallas ibland också processor). Kan du beskriva vilken funktion den har?
7. Försök ta reda på vilken CPU som sitter i din dator. Är den av familjen Intell, AMD eller något annat? Vilken klockhastighet har den?
8. Ett operativsystem består av en stor uppsättning program.
    a. Kan du nämna några funktioner operativsystemet har?
    b. Kan du räkna upp några olika operativsystem?
    c. Vilket operativsystem använder du? (Namn, version)
9. Programmeringsspråk är till för människor, eftersom det är lättare för oss att uttrycka oss i ett programmeringsspråk än i ett format som din dator direkt kan förstå. Kompilerade språk, såsom C och C++, använder en *kompilator* för att skapa ett körbart program från källkod skrivet i programmeringsspråket. Försök beskriva varför ett program skrivet i något programmeringsspråk och som är kompilerat på en macOS-dator inte går att köra i en dator med Windows (utan att kompilera om för Windows).
10. Använd en sökmotor för att ta reda på vem Grace Hopper var. Kan du nämna något av hennes bidrag till utvecklingen av program och programmering?

Filer, kataloger och sökvägar

1. Vilken är en vanlig abstraktion för hur vi organiserar filer i ett filsystem?
    a. Vi tänker på en abstraktion för behållare av filer och fler behållare
2. Vilken är fördelen med att inte ha alla filer “i en hög” eller i en lång rad?
3. Vad kallas den katalog som man brukar rita ut (eller säga ligger) högst upp i ett katalogträd?
    a. Vad har den för beteckning på din dator?
4. Vilka två typer av sökvägar (eng: *path*) finns det? (Ledtråd: den ena utgår från toppen av filträdet och den andra från en viss katalog)
    a. Vad är det för skillnad på dem?
    b. När kan man använda den ena sorten och när kan man använda den andra?
5. Vad används symbolerna **.** respektive **..** till (punkt resp. punkt-punkt)?
6. Tror du man kan arbeta i en terminal utan att lära sig hur sökvägar fungerar?
    a. Varför/varför inte?
7. Vad menas med *aktuell katalog (eng.: current directory)*?
8. Vad menas med *hemkatalog (eng.: home directory, home)*?

Terminalmiljö, kommandon, navigera i filträdet, och lite blandat

1. Vad används en terminal till?
2. Vilken roll eller funktion har en kommandotolk (också kallat ett skal)?
3. Kompendiet fokuserar på skalet Bash. Har du Bash installerat på din dator?
    a. Om du inte har det och kör Windows, installera Cygwin eller Ubuntu for windows för att kunna göra övningarna i detta häfte!
4. Vad är ett kommando?
    a. Ge exempel på några kommandon och beskriv vad de gör
    b. Vad är ett argument till ett kommando? Hur ger man argument? Ge exempel på kommandon som tar argument och förklara vad argumentet står för.
    c. Kan alla kommandon acceptera argument? Om inte, ge exempel på kommandon som inte kan ta några argument och förklara varför.
5. Vad är kommandoprompten?
    a. Hur ser kommandoprompten ut i din terminal (som kör bash)
    b. Vad består kommandoprompten av för information?
6. Beskriv begreppet “aktuell katalog”
    a. Hur byter man aktuell katalog?
    b. Vilket “smeknamn” eller kortform har aktuell katalog?
    c. Ge exempel på hur man kan använda kortformen för aktuell katalog som ett argument, till exempel vid kopiering av fil, listning av aktuell katalog
7. Beskriv begreppet “parent directory” eller “katalogen ovanför”.
    a. Vilket “smeknamn” eller kortform har “parent directory”?
    b. Ge exempel på hur denna kortform kan användas som argument till ett kommando, till exempel för att kopiera fil eller byta katalog till katalogen ovanför.
8. Beskriv begreppet “hemkatalog”.
    a. Vilket “smeknamn” eller kortform har aktuell användares hemkatalog?
    b. Ge exempel på hur denna kortform kan användas som argument till kommandon, till exempel vid kopiering av en fil till hemkatalogen eller byte av aktuell katalog till hemkatalogen.
    c. Vilken sökväg har din användares hemkatalog, enligt bash?
        i. Hint: använd kommandot `pwd` när du står i din hemkatalog
        ii. Hint: du kan också använda kommandot `echo` och ge kortformen för hemkatalogen som argument till `echo`
        iii. Hint: du kan också använda kommandot `echo` och miljövariabeln `$HOME`
    d. Prova att använda `echo` och kortformen för hemkatalogen omeddelbart följt av ditt användarnamn. Vilken hemkatalogs sökväg skrivs ut?
        i. Vad tror du skulle hända om du gjorde samma sak men angav något annat användarnamn direkt efter kortformen för hemkatalogen
    e. Vilken katalog är aktuell katalog när du startar din terminal? Varför tror du det är så?

    f. Vilken katalog hamnar du i om du ger kommandot `cd` utan argument?
        i. Hint: Prova!
9. En speciell sorts argument till kommandon kallas för “options” eller “flaggor” eller “växlar”. Vilket tecken brukar de börja med?
    a. Vilken flagga ska du ge till kommandot `ls` för att få en längre listning av filer och kataloger?
    b. Vilken flagga ska du ge till kommandot `ls` för att få med även “dolda filer”?
    c. Vad avgör ifall en fil eller katalog i Bash är “dold” dvs inte listas av `ls` utan flaggan vi frågar efter ovan.
10. En lång listning av en fil kan se ut såhär:
    ```
    -rw-rw-r-- 1 rikard rikard 103 sep  6 17:41 Hello.java
    ```
    a. Beskriv den del som ser ut så här: `-rw-rw-r--`
    b. Vad betyder de två “`rikard`”?
    c. Vad är `103`?
    d. När ändrades filen senast?
    e. Vad heter filen?
11. Prova att ge Bash följande kommandorad i din terminal:
    ```
    echo "Filerna:"; ls; echo "Såg det rätt ut?"
    ```
12. Använd kommandot uname för att ta reda på ditt operativsystems namn och version.
13. Öppna en terminal och ta reda på den absoluta sökvägen till aktuell katalog. Skapa följande katalogträd:
    `courses/tig015/it/working-in-the-terminal`
    Du kan välja om du vill skapa trädet med fyra kommandon med `mkdir` eller om du vill skapa hela trädet med ett kommando (Hint: läs på om flaggan `-p`).
14. Gå ned i katalogen `working-in-the-terminal` i trädet du just skapade. Vilken katalog hamnar du i om du ger kommandot:
    ```
    cd ../..
    ```
    ?
15. Gå tillbaka till katalogen `working-in-the-terminal` (hint: du kan använda kommandot `cd -` för att gå tillbaka till “förra katalogen”; ett ensamt bindestreck som argument till `cd` betyder just “förra katalogen”). Använd en absolut sökväg för att gå till katalogen `tig015` i trädet. Läs på om absoluta och relativa sökvägar om du glömt bort hur man gör.
16. Om du står i katalogen `courses` som ligger i din hemkatalog, vilken är den relativa sökvägen till `working-in-the-terminal` ? Vilken är den absoluta sökvägen till samma katalog?
17. Undersök vilka standardkataloger i din hemkatalog du fått vid installation av ditt operativsystem. Hint: Kataloger du inte skapat själv men som ändå ligger i din hemkatalog är sådana kataloger.
18. Undersök vilka dolda filer som finns i din hemkatalog med hjälp av kommandot `ls` och lämplig flagga.
19. Gå ned i katalogen `courses` och utfärda kommandot `tree` . Överst i trädet är en punkt. Vilket namn har katalogen som punkt refererar till? Vilken absolut sökväg har den katalogen?

20. Fortfarande i katalogen courses, skriv (utan att trycka Enter):
    `ls t`
    och tryck sedan på Tab (tabulatorknappen). Vad händer?
21. Fortfarande i katalogen `courses`, skapa katalogen `tig058` . Kör sedan kommandot
    `ls -l`
    Vilka filer listas? Vilka av dem är kataloger? Hur vet du det?
22. Fortfarande i katalogen `courses`, skriv igen (utan att trycka Enter):
    `ls t`
    och tryck sedan på Tab. Vad händer? Tryck på Tab igen. Vad händer nu? Lägg till tecknet `5` och tryck på Tab igen. Vad händer? Försök förklara hur "Tab-expansion" fungerar.

     Tab-tangentens symbol på tangentbordet.

23. Gå till din hemkatalog med hjälp av kommandot `cd` . Skapa följande katalogträd:
    `a/whole/lotta/directories`
24. Ta bort hela trädet som börjar med katalogen `a/` . Hur många kommandon behöver du använda om du inte får använda några flaggor till `rmdir` ? I vilken ordning måste du utföra kommandona? Varför? Vilken flagga kan du använda för att ta bort `a/` och alla dess underkataloger (om de är tomma)?
25. Utan att lämna din hemkatalog, öppna och skapa en textfil med den relativa sökvägen
    `courses/tig015/it/working-in-the-terminal/notes.txt`
    Använd en editor för att göra detta. Här är exempel på editorer:
    atom
    nano
    vim
    emacs
    Notepad++
    Sublime Text
    Visual Studio Code
26. Läs i kompendiet för hur du kan skapa textfilen med emacs och använd emacs för detta. Lägg in Följande text i filen:
    `One line of text.`
27. Spara filen och gå ur emacs. (Hint: Ctrl-x Ctrl-s sparar, Ctrl-x Ctrl-c stänger emacs.)
28. Skriv ut innehållet i filen med hjälp av kommandot `cat` utan att lämna din hemkatalog (Hint: du måste ange hela sökvägen till filen, lättast som en relativ sökväg).
29. Kör följande kommando med sökvägen till din fil som argument på slutet:
    `hexdump -v -e '/1 "%03i "'`
    Utskriften innehåller de decimala värdena i ASCII-tabellen för texten i filen. Vilket värde har mellanslag? Kan du se det värdet i utskriften? Hur många gånger?

Enkla Bash-skript

1. Skapa följande katalog `courses/tig015/it/bash-script`
2. Använd en editor för att skapa följande skript:
   ```
   #!/bin/bash
   echo -n "Time in Tehran is now "
   TZ='Asia/Tehran' date +%T
   ```
3. Döp skriptet till `tehran_time.sh`
   Hint: Det enklaste är att ge din editor följande argument (när du står i din hemkatalog):
   `courses/tig015/it/bash-script/tehran_time.sh`
   eftersom filen skapas på rätt ställe då när du sparar.
4. Ändra rättigheter på filen så att du får “execute”-rättigheter. Läs på i kompendiet hur du använder `chmod` för att uppnå detta.
5. Kör skriptet från din hemkatalog.
6. Skapa en katalog i din hemkatalog med namnet `bin/`
7. Flytta skriptet till denna katalog (med kommandot `mv`)
8. Öppna filen `.bashrc` som ligger i din hemkatalog (eller skapa den om du saknar en sådan fil).
9. Om du hade filen, leta upp en rad som innehåller variabeln PATH . Om du skapar filen för att den saknades, skapa följande sats:
   `PATH="$PATH:/home/ditt-användar-namn/bin"`
   Om du hade filen och hittade en rad som sätter PATH-variabeln, lägg till följande på slutet:
   `:/home/ditt-användar-namn/bin`
   Lägg annars till samma rad som anges ovan:
   `PATH="$PATH:/home/ditt-användar-namn/bin"`
10. Det du gjorde ovan var att lägga till din nya katalog `bin/` i din PATH-variabel, vilken är den variabel som innehåller sökvägar med kataloger där Bash kan leta efter kommandon du skriver i kommandoraden. Nu kan du köra ditt nya skript utan att ange sökväg till skriptet. Men först måste du läsa in .bashrc på nytt (vilket sker automatiskt om du öppnar en ny terminal):
    `. .bashrc`
11. Kommandoraden ovan börjar med en punkt. Det är ett inbyggt kommando i Bash faktiskt som betyder “läs in”. Du instruerade just Bash att läsa in .bashrc vilket gjorde att din variabel PATH nu innehåller din nya katalog `bin/` .
12. Kör ditt nya skript genom att endast ange skriptets namn (och inte sökvägen till skriptet).

Ladda ned filer från webben

1. Skapa en katalog med ett passande namn för att lagra textfiler i
2. Gå ned i katalogen
3. Använd wget för att ladda ned följande filer till katalogen:

    a. https://raw.githubusercontent.com/progund/intro-it/master/text-processing/meals.txt
    b. https://raw.githubusercontent.com/progund/intro-it/master/network-protocols-data/workshop/pic.html
    c. https://raw.githubusercontent.com/progund/intro-it/master/bash-scripting-intro/exercises/datetime.sh
    d. https://github.com/progund/intro-it/raw/master/bash-scripting-intro/exercises/curriculum-tig015.pdf
    e. https://raw.githubusercontent.com/progund/intro-it/master/README.md
4. Använd kommandot `file` för att ta reda på vad för filtyp filerna har (enligt `file`)
5. Använd ett lämpligt program för att öppna filerna från terminalen
6. Radera filerna med kommandot `rm`
7. Ladda ned filerna igen, denna gång med hjälp av kommandot `curl`
    a. Läs kompendiet och wikin för att hitta rätt syntax för kommandot
8. Besök denna sida i en webbläsare: https://github.com/progund/intro-it/blob/master/bash-scripting-intro/welcome.sh
9. Ett vanligt misstag är att använda samma URL för att ladda ned skriptet. Använd `wget` eller `curl` för att ladda ned sidan med ovanstående url
10. Använd `ls` för att se om något laddades ned (finns filen `welcome.sh` ?)
11. Använd `cat` för att skriva ut skriptet i terminalen - ser det rätt ut?
12. Använd `file` för att ta reda på vad för filtyp filen du laddade ned hade
13. Kan du komma på vad som gick fel?
    a. Ledtråd: gå till sidan i din webbläsare igen och klicka på knappen med texten “Raw” i raden ovanför skriptet - hur ser den sidan ut? Hur ser den URL:en ut?
    b. Ledtråd: radera den gamla filen (som såg konstig ut när du skrev ut den)
    c. Ledtråd: ladda ned skriptet med hjälp av den nya URL:en (efter att du klickade på Raw, kom du till en ny URL)
    d. Ledtråd: använd `file` för att undersöka filen igen - vad säger `file` nu?
    e. Ledtråd: använd `cat` för att skriva ut filen i terminalen igen - ser det rätt ut nu?
14. Bonusuppgift:
    a. När du fått ned rätt version av welcome.sh, skriv följande (dollartecknet föreställer prompten och ska inte skrivas)
    ```
    $ bash welcome.sh
    ```
    b. Utför följande kommando:
    ```
    $ ls -l welcome.sh
    ```
    hur ser rättigheterna för filen ut?
    c. Utför följande kommandon:
    ```
    $ chmod u+x welcome.sh
    $ ls -l welcome.sh
    ```
    Vad har ändrats i rättigheterna?
    d. Utför följande kommando:
    ```
    $ ./welcome.sh
    ```
    Vad hände? Kan du beskriva vad du gjort för att kunna köra skriptet på detta vis?

Skapa filer från utskrifter från kommandon

1. Gå till katalogen där du laddade ned filer
2. Skapa en fil som heter `file-list.txt` med hjälp av kommandot ls och omdirigering av standard out
    a. Ledtråd: för att omdirigera std out från ett kommando till en fil, kör:
    `$ kommando > resultat-fil.txt`
3. Använd `cat` för att skriva ut innehållet i file-list.txt
4. Är filen själv, `file-list.txt`, med i listan?
    a. I så fall, vad säger det om när filen skapades i relation till när kommandot `ls` kördes?
5. Skriv över filen med nytt innehåll genom att i stället omdirigera följande kommandos std out till filen:
    `$ ls -1`
    Flaggan är en etta, inte lilla L!
6. Använd `cat` igen för att se innehållet i terminalen. Hur ser det ut denna gång?
7. Skriv över filen igen, denna gång med omdirigering av kommandot:
    `$ ls *.txt`
8. Använd *append* av standard out (tips: det var som vanlig omdirigering med med tecknet två gånger och append betyder skriv till slutet av filen, skriv inte över) för att fylla på filen med append av följande kommando:
    `$ ls *.html`
9. Använd *append* igen och kommandot
    `$ ls *.pdf`
    och igen med kommandot
    `$ ls *.md`
    och igen med kommandot
    `$ ls *.sh`
10. Använd `cat` för att se innehållet i filen. Hur är filen ordnad (sorterad) denna gång? Jämför med att bara köra `ls` som vanligt i terminalen utan omdirigering.
11. Nu ska vi visa hur du kan skapa en textfil och skriva in text i den utan att använda en editor. Följ anvisningarna till pricka, så lär du dig ett sätt att snabbt skapa mindre textfiler.
12. Skriv följande och tryck Enter (som vanligt är $ prompten och ska inte skrivas av dig):
    `$ cat > my-file.txt`
13. Lägg märke till att du inte fick en ny prompt! Nu körs cat och läser från sin standard input-ström (som råkar vara kopplad till ditt tangentbord, via terminalen).
14. Skriv in En rad text med valfritt innehåll och tryck på Enter och en rad till med text avslutat med Enter.
15. Tryck Enter igen, så att du kan se att det är ett nytt stycke (en tom rad mellan förra raderna och nya texten) och skriv in en rad till och tryck Enter.
16. Tryck `Ctrl-D` (kontrolltangenten och samtidigt d-tangenten)
17. Lägg märke till att du fick tillbaka Bash-prompten, det vill säga `cat` är färdig och du är tillbaka i kommandotolken Bash.

18. Använd `cat` igen för att skriva ut filen i terminalen (som vanligt, ingen omdirigering).
19. Om du gjort rätt så ser du nu din nya text som du skapade med hjälp av `cat` och omdirigering av std out till fil.
20. När du tryckte `Ctrl-D`, så skickade terminalen en signal till `cat` som sa att här kommer ingen mer data, så cat slutade läsa från std-in (tangentbordet i detta fall) och terminerade (blev klar, avslutades).

Arbeta med textfiler

1. Först behöver du skapa en ny katalog med lämpligt namn för dessa uppgifter. Kanske `text-processing` eller `text-commands` eller `text-exercises` kan vara lämpligt?
2. Gå ned i den nya katalogen, så att du kan arbeta ostört och utan massa gamla filer som ligger ivägen.
3. Nu behöver du lite textfiler. Du vet ju nu hur du kan ladda ned filer från webben till aktuell katalog med wget eller curl, så det ska inte vara några problem. Ta ned dessa filer:
    a. https://github.com/progund/databases-introduction/raw/master/scraping-data/examples/teaching/tig015h19.csv
    b. https://github.com/progund/databases-introduction/blob/master/workshop-1/banned_new.txt
    c. https://github.com/progund/datorkunskap-kompendium/raw/master/text/replaceme.txt
    d. https://github.com/progund/datorkunskap-kompendium/raw/master/text/a_few_urls.txt
    e. https://github.com/progund/datorkunskap-kompendium/raw/master/text/urls.txt
4. Den första filen är en CSV (comma-separated values) med schemat för en kurs på universitetet i Göteborg.
5. Den andra filen är en lista på förbjudna teckenkombinationer (enligt Transportstyrelsen) för registreringsnummer
6. Den tredje filen är en kort textfil med felstavningar
7. Den fjärde filen är en textfil med några URLer.
8. Den femte filen är en textfil med massor av URLer.
9. Nedan kommer övningarna.

Analysera schemat

Börja med att ladda ned https://github.com/progund/databases-introduction/raw/master/scraping-data/examples/teaching/tig015h19.csv ifall du inte redan gjort det.

Vi börjar med att skriva ut första raden i denna CSV-fil, för att se vad rubrikerna säger att varje kolumn (separerade med komma) betyder.

Använd `head` för att skriva ut endast första raden. Vilken kolumn (nummer) innehåller lärare?
Tips: `head -5` tar de *fem* första raderna i en fil. Så den första raden blir...?

Svar: den nionde kolumnen innehåller lärare.

Använd nu `cut` med omdirigering av std in att läsa från filen för att splitta fälten på kommatecken och enbart skriva ut fält nummer 9.

Tips: för att köra `cut` med läsning från fil i stället för från standard in, använd
`$ cut flaggor < filnamn`

Tips: flaggan för att välja avskiljare för fälten är `-d` och sätt tecknet (kommatecken) inom enkelfnuttar efter `-d` . Flaggan för att välja ut endast fält nummer `N` är `-fN` där N är ett heltal, t ex 9 .

Nu vill vi se vilka lärare som finns i kursen. Detta kan vi göra genom att skriva ut kolumn 9 igen och sortera resultatet unikt. Vi kommer använda en pipe (ett vertikalt streck) för att koppla ihop output from cut till input för sort. Du kommer lära dig mer om pipelines (att använda pipe på detta sätt) om du läser kompendiet och på wikin, men skriv av så länge och försök hänga med.

Här är ett förslag som du ska testa:
```
$ cut -d ',' -f9 < tig015h19.csv | sort -u
Aida Hadzic Zukic
"Aida Hadzic Zukic;Anna-Lena Fredriksson;Kalevi Pessi;Urban Nuldén"
"Aida Hadzic Zukic;Henrik Sandklef;Kalevi Pessi;Rikard Fröberg"
"Aida Hadzic Zukic;Kalevi Pessi"
"Aida Hadzic Zukic;Kalevi Pessi;Urban Nuldén"
"Henrik Sandklef;Rikard Fröberg"
Kalevi Pessi
Lärare
```

Hm. Nu kom ju kolumn 9 med från rubrikraden, “Lärare” också. Hur kan vi få bort detta? Det finns många sätt, men ett sätt är att stoppa in:
| grep -v Lärare
någonstans i pipelinen, till exempel så här:
```
$ cut -d ',' -f9 < tig015h19.csv | grep -v Lärare | sort -u
Aida Hadzic Zukic
"Aida Hadzic Zukic;Anna-Lena Fredriksson;Kalevi Pessi;Urban Nuldén"
"Aida Hadzic Zukic;Henrik Sandklef;Kalevi Pessi;Rikard Fröberg"
"Aida Hadzic Zukic;Kalevi Pessi"
"Aida Hadzic Zukic;Kalevi Pessi;Urban Nuldén"
"Henrik Sandklef;Rikard Fröberg"
Kalevi Pessi
```

Ett annat “problem” är ju att lärarna i själva verket ju är kombinationer av lärare där det är ett semikolon mellan lärare i en kombination med mer än en lärare...

Här är ett förslag på hur vi löser detta:

Kör raden ovan igen men lägg till på slutet att alla semikolon ska transponeras (översättas eller ersättas) till nyradstecken (\n kan man använda för nyrad). Om du vill försöka själv, så använd `tr` för att transponera/ersätta. Men här är vårt förslag:

Vi provar:

```
$ cut -d ',' -f9 < tig015h19.csv | grep -v Lärare | tr ';' '\n' | sort -u
"Aida Hadzic Zukic
Aida Hadzic Zukic
Anna-Lena Fredriksson
"Henrik Sandklef
Henrik Sandklef
Kalevi Pessi
Kalevi Pessi"
Rikard Fröberg"
Urban Nuldén"
```

Problemet nu är citationstecknen, som ju får samma lärare att räknas olika beroende på om det finns citationstecken i raden eller inte. Så vi tar bort dessa genom att använda tr igen med flaggan -d (delete):

```
$ cut -d ',' -f9 < tig015h19.csv | grep -v Lärare | tr ';' '\n' | tr -d '"' | sort -u
Aida Hadzic Zukic
Anna-Lena Fredriksson
Henrik Sandklef
Kalevi Pessi
Rikard Fröberg
Urban Nuldén
```

Vi tittar på pipelinens beståndsdelar och förklarar dem en och en:

```
cut -d ',' -f9 < tig015h19.csv
# skriv ut enbart kolumn nio från filen (använd komma som separator)

| grep -v Lärare
# från detta resultat, använd grep för att bortse från rader som
# innehåller Lärare

| tr ';' '\n'
# från detta resultat, byt ut alla semikolon mot nyradstecken

| tr -d '"'
# från detta resultat, radera alla citationstecken

| sort -u
# Från detta resultat, sortera raderna och ta bort dubbletter
```

Låt oss nu undersöka vilken undervisningstyp som är vanligast. Undervisningstyp är kolumn nummer 10.

Strategi: Skriv ut alla kolumn 10 från filen medelst `cut`, använd pipeline för att sortera medelst `sort`, sedan ta bort dubbletter och räkna dem nedelst `uniq -c`, sedan sortera numeriskt fallande med avseende på första kolumnen medelst `sort -nrk1` .

Prova först själv innan du läser vidare vårt exempel på lösning. Som vanligt när du tar fram en pipeline (tycker vi) kan du köra steg-för-steg och lägga på filter med pipe för att komma närmare resultatet.

Vårt lösningsförslag (som går att göra kortare och på oändligt många sätt) finns i slutet på övningshäftet. Försök själv först!

Manipulera URLer

Nästa övning gäller filen a_few_urls.txt

Nu vill vi bearbeta filen på olika sätt för att få ut delar av URLerna och sortera på olika sätt.

Vi börjar med något enkelt, så som att klippa bort protokolldelen av URLerna (http:// eller https://).

Använd cut för att separera fälten på slash (snedstreck framåt) och skriva ut fält 3 och resten.

Tips: `cut -d` väljer separator, `-f` väljer fält. Om du vill ha fält 3 och resten, använd `-f3-` där bindestrecket betyder “osv”.

På detta vis kan du dela upp en url i fälten:

```
http:    /    /    www.osvosv...../directory/file...
  1        2             3             4      5 osv
```

Det som kan vara knepigt att förstå initialt är att om du använder slash som separator och det förekommer två slash i rad (som i http://) så är det faktiskt ett tomt fält (nr 2) mellan slasharna. I slutet av övningshäftet finns lösningsförslag.

Försök nu filtrera ut enbart domännamnet från URLerna. Det torde vara ganska enkelt om du klarade första uppgiften. Tips: om du vill ha enbart ett visst fält, ange endast fältnummer (och inget bindestreck efteråt, eftersom du inte vill säga “osv”). Lösningsförslag finns i slutet.

Nästa utmaning blir att hitta URL:er som slutar med ett filnamn som innehåller en punkt och en filändelse (t ex `.html`), till exempel filen `afterword_v1a_singlefile.html` .

Du får använda grep och ett reguljärt uttryck (regular expression).

Tips: Tänk så här:
Rader med vad som helst följt av en verklig punkt, följt av bokstäver a-z följt av radslut.
Tips: Att uttrycka “vad som helst” kan göras med **`.*`** där punkt står för “vilket tecken som helst” och stjärnan står för “noll eller flera”.
Tips: Att uttrycka “bokstäver a-z” kan göras med **`[a-z]`**
Tips: Radslut betyder att inget mer får komma efter förra delen av uttrycket och kan uttryckas med **`$`** .
Kika i lösningsförslaget om du inte får till det och försök förstå lösningen - det är nyttigt att åtminstone kunna läsa och förstå reguljära uttryck, även om man inte alltid kan formulera dem själv. Man lär sig läsa innan man lär sig skriva, och det gäller bash och programmering också.

Nästa utmaning med URLerna är att filtrera ut enbart domännamnen och gruppera alla domännamn med avseende på toppdomän (toppdomän är t ex .se .edu .org .com osv). Det vill säga, vi vill skriva ut enbart domänerna igen men alla .se i följd och alla .org i följd osv. Hur ska vi kunna göra det?

Förväntad output:

```
ait.gu.se
www.gu.se
www.elsewhere.org
snarxiv.org
www.physics.nyu.edu
```

Tips: Använd `rev` för att skriva ut varje rad baklänges och sortera detta. Sedan `rev` igen för att få raderna framlänges igen:

```
es.ug.www
es.ug.tia
gro.erehwesle.www
gro.vixrans
ude.uyn.scisyhp.www
```

Fånga upp ovanstående i en pipe och sortera det (nu råkar det redan vara sorterat men det är en slump). Fånga upp det sorterade i en pipe och kör `rev` igen så att de skrivs ut grupperat på toppdomän.

Se slutet av övningshäftet för ett lösningsförslag.

Att redigera kommandoraden samt en del tricks

Sista kommandots sista ord

Skapa en katalog i din hemkatalog som heter `test` . Du ska använda `mkdir` för detta:

```
$ mkdir test
```

Innan du gör något annat, skriv följande på nästa kommandorad:

```
$ cd <Esc .>
```

(skriv cd tryck sedan en gång på Escape tangenten och sedan på punkttangenten).

Bash fyllde i `test` åt dig! Tryck Enter för att utföra kommandot.

Gå tillbaka där du stod genom

```
test$ cd ..
```

Ta bort katalogen med `rmdir test` så du får bort testkatalogen.

Leta fram mkdir-raden från din Bash history genom att trycka Ctrl-R (control-tangenten och samtidigt r-tangenten) och skriv `mk`
Så här ser det ut i din prompt:

```
(reverse-i-search)`mk': mkdir test
```

Som du ser, hittade bash-historiken raden med hela `mkdir test` åt dig, trots att du bara skrev `mk` !

Innan du gör något annat, tryck Ctrl-E för att välja raden ur historien och gå till slutet av raden. Nu ser din prompt ut så här:

```
$ mkdir test
```

och du har markören på slutet. Tryck nu Ctrl-W (control-tangenten och samtidigt w-tangenten) för att sudda ut ordet till vänster om markören. Din kommandorad ser nu ut så här:

```
$ mkdir
```

Skriv nu `test2` och tryck Enter.

Reflektion:
Du skapade först en katalog som hette test. För att gå ned i katalogen använde du Esc+. (först Esc och *sedan* punkt) för att fylla på din påbörjade rad med cd med förra radens sista ord (test).

Sedan gick du upp igen och tog bort test. Du kom på att du ville skapa en katalog till och sökte fram din `mkdir test` ur Bash-history med Ctrl-R (control och *samtidigt* r) genom att endast ange mk . Sedan flyttade du dig till slutet av raden, raderade sista ordet genom Ctrl-W och skrev ett nytt namn i stället.

Detta vis att redigera kommandoraden och återanvända gamla kommandon eller delar av gamla kommandon är väldigt vanligt hos vana terminalanvändare. Vill du bli effektiv, så är det vårt tips att träna på liknande saker. Hitta på lite egna övningar!

Läs kompendiet och wikin och testa följande kortkommandon:

- `Ctrl-A` - flytta till starten av raden
- `Ctrl-E` - flytta till slutet av raden
- `Ctrl-←` - flytta ett ord åt vänster
- `Ctrl-→` - flytta ett ord åt höger
- `Ctrl-K` - radera från markören och resten av raden
- `Ctrl-W` - radera ett ord (i taget) åt vänster
- `Ctrl-Y` - klistra in senaste raderade text

Gick det bra?

Nu ska vi testa hur exit status fungerar. Först genom att undersöka shell-variabeln $?

Skriv `ls` och namnet på en fil eller katalog som du vet inte finns. Skriv som nästa kommando `echo $?` . Exempel:

```
$ ls apa
ls: cannot access 'apa': No such file or directory
$ echo $?
2
$
```

Skriv nu ls och namnet på en fil du vet finns. Och så echo $? igen.

```
$ ls urls.txt
urls.txt
$ echo $?
0
$
```

Av detta förstår vi att när ett kommando misslyckas eller går fel, så sätts `$?` till ett värde skiljt från noll och när ett kommando lyckas (går bra) så sätts `$?` till noll. Exakt vad siffror som inte är noll betyder varierar från kommando till kommando och vi får läsa manualen för att förstå varför.

För att läsa manualen till exempelvis `ls` så skriver du `man ls` för att komma in i manualläsaren. Du kan skrolla upp och ned med piltangenterna, använda space för att hoppa ned en hel sida, söka efter ord med / och för att komma ur manualläsaren trycker du på q-tangenten.

Här är ett stycke ur manualen för `ls`:

```
   Exit status:
       0      if OK,
       1      if minor problems (e.g., cannot access subdirectory),
       2      if serious trouble (e.g., cannot access command-line argument).
```

Och här från `grep`:

```
EXIT STATUS
  Normally the exit status is 0 if a line is selected,
   1 if no lines were selected,  and
   2 if  an error  occurred.
  However,  if the -q or --quiet or --silent is used and a line is
  selected, the exit status is 0 even if an error occurred.
```

Det alla välskrivna kommandon har gemensamt är att noll betyder “success” eller “normalt avslut” (inga fel eller problem).

Som du läst i kompendiet och i wikin, så kan vi utnyttja detta för att köra två kommandon beroende på hur det första rapporterar att det gick:

```
$ which superkommando || echo "superkommando could not be found"
superkommando could not be found
$ which ls && echo "The ls command could be found."
/bin/ls
The ls command could be found.
```

Kommandot which kollar om ett kommando finns i PATH-variabelns kataloger (går att hitta och därför köra):

```
$ which ls
/bin/ls
$ which grep
/bin/grep
$ which wget
/usr/bin/wget
$ which which
/usr/bin/which
```

Om det inte hittas, skrivs ingenting men $? sätts till 1 (ett):

```
$ which witch
$ echo $?
1
```

Man kan till och med använda IF-satsen som har följande struktur:
`if kommando; then kommando; else kommando; fi`
Till exempel:

```
$ if ls urls.txt; then echo "The file urls.txt could be listed"; else echo
"No urls.txt file found";fi
urls.txt
The file urls.txt could be listed
$ if ls money.txt; then echo "The file money.txt could be listed"; else
echo "No money.txt file found";fi
ls: cannot access 'money.txt': No such file or directory
No money.txt file found
```

Det kanske blir tydligare om vi omdirigerar std error till slukhålet /dev/null:

```
$ if ls money.txt 2> /dev/null; then echo "The file money.txt could be
listed"; else echo "No money.txt file found";fi
No money.txt file found
```

Skriv en kommandorad som om kommandot wget är installerat hämtar en fil och annars skriver ett felmeddelande. Använd först en kombination av && och || sedan if-else.

Globbing och expansion

Lista filer med glob-uttryck

Lista alla filer, i en katalog med många filer, vars namn slutar på "**`.txt`**" . Lista alla filer som börjar på en av två bokstäver, t ex "**a**" eller "**u**" - använd klamrarna från globbing-syntaxen för att skapa en lista med de två bokstäverna.

Lista alla filer som börjar på en av de två bokstäverna igen och som dessutom har namn som slutar på **`.txt`** .

Lista alla filer som *inte* börjar på a eller u.

Använd brace expansion { }

Använd *en* echo-sats för att med hjälp av *brace expansion* skriva ut följande text:
`ap*skaft* ap*anage* ap*-fonden* ap*elsin* ap*pell* ap*a* ap*parat*`

Använd *en* echo-sats för att med hjälp av brace expansion skriva ut 20 ord, grupp1 till grupp20:
`grupp1 grupp2 grupp3 ... grupp20`

Använd en echo sats (med brace exp.) för att skriva ut följande ord (i följande ordning):
`grupp20 grupp15 grupp10 grupp5`
Se lösningsförslag i slutet på häftet.

Variabler

Skriv ut värdet på din miljövariabel $PATH i terminalen. Öppna ytterligare en terminal (utan att stänga den första) och skriv ut variabeln där. Blev det samma värde?

Ha kvar den nya terminalen men byt fokus till den första. Skapa en variabel med namn `BACKUP_DIR` och värdet `"$HOME/backups"` i den första terminalen.

Hint: Så här skapar man en variabel med namn **`A`** och värdet **`"apa"`**:
```
$ A="apa"
$
```
Byt fokus till den andra terminalen och skriv:
```
echo "$BACKUP_DIR"
```

Skrevs något ut?

Byt till baka till första terminalen, den där du skapade variabeln. Skriv samma echo-sats igen. Skrevs något ut?

Poängen med denna enkla övning är att visa att miljövariabler, som t ex **`$PATH`**, har samma värde i alla nya skal (bash-instanser) om du inte sätter om dem explicit i ett av skalen.

En variabel du själv skapar i ett skal finns inte i nästa skal du startar. Det som hände var att du körde en terminal med Bash från början. I den sessionen med Bash satte du en variabel. När du öppnade en terminal till, så startar en ny instans av Bash. I denna nya instans finns inte den variabel du satte i första terminalen. Bara “miljövariabler”, såsom `$PATH`, existerar i båda instanserna av Bash. Anledningen är faktiskt helt omystisk. En miljövariabel, skapas och får sitt värde i en fil som Bash läser varje gång Bash startar. Så varje ny instans av Bash har läst in denna fil och skapat och satt värde på samma variabler - så kallade miljövariabler.

Ett exempel på en sådan inställningsfil finner du i /etc/environment (på Ubuntu i vart fall). Skriv följande kommando (i vilken terminal som helst):
```
$ grep PATH /etc/environment
```

På Rikards dator blev resultatet följande:
```
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/
usr/games:/usr/local/games"
```

Poängen är i vart fall att Bash läste in en fil vid uppstart och skapade variabeln och dess värde. Det är så kallade miljövariabler (eng: environment variables).

Det finns ett sätt att skapa en variabel som en ny instans av Bash känner till. Detta under förutsättning att du från Bash startar en till instans av Bash.

Om du i ditt skal skriver `bash [Enter]` så startar du Bash från Bash. För att logga ut från den nya Bash-instansen trycker du `Ctrl-D` . En Bash-instans som skapas från en annan Bash-instans kallas *barninstans* eller *barnprocess* (eng: child process). Den bash-process som kör i ditt skal när du startar Bash därifrån blir då analogt *föräldraprocess* (eng: parent process).

Du kan *exportera* en variabel till alla barnprocesser genom att skriva **`export`** före variabeldeklarationen. Då finns variabeln i alla Bash som startas där nuvarande Bash-process är en föräldraprocess (eller faktiskt förfäder-process).

Detta är inte jätteviktigt att kunna men kan vara bra att ha hört om du stöter på något förvånande gällande variabler. Ett kort exempel:

I din terminal, skriv:
```
$ export BIRD="Bluebird"
```

Starta nu Bash på nästa rad och skriv ut variabeln `BIRD`:
```
$ bash
$ echo $BIRD
Bluebird
```
Logga ut från nya instansen av Bash genom `Ctrl-D` .

Skapa nu en ny variabel `FLOWER` med värdet `"Rose"` utan `export` före:
```
$ FLOWER="Rose"
```

Starta en ny instans av Bash från terminalen och skriv ut värdet av `FLOWER`:
```
$ bash
$ echo $FLOWER

$
```
Som du såt, så kände nya (child process) Bash inte till variabeln FLOWER, eftersom den inte var *exporterad*.

Återigen, detta är inte livsviktigt att känna till men ibland stöter man på situationer då man blir förvånad över att en variabel plötsligt är “tom”. Det beror oftast på att man startat en child process och inte exporterat variabeln.

Förresten, det är inte möjligt att exportera en variabel från en child process till en föräldraprocess. Det vill säga om du är i Bash och startar en ny instans av Bash och där sätter en variabel och sedan loggar ut, så “försvinner” den variabeln. Det är inte så konstigt om man tänker på att en process inte får påverka en annan process som redan kör. Men en process som startar en annan process kan vid starten “skicka med” lite information (så som exporterade variabler).

Uppgift:

Skapa tre variabler: SECS_PER_MIN, MINS_PER_HOUR, HOURS_PER_DAY och tilldela dem lämpliga värden. Skriv ut med `echo` en rad som använder variablerna, t ex:
`Det är 60 sekunder på en minut, 60 minuter på en timme och 24 timmar per dygn.`

Där siffrorna alltså kommer från variablerna.

Kommandosubstitution

Eftersom de flesta kommandon i Bash genererar något slags textutskrift (eng: text output), så vore det smidigt om man kunde använda ett kommando som en del av utskriften från ett skript eller en echo-sats, eller om man kunde använda utskriften från ett kommando som en del av argumenten till ett annat kommando.

Detta är faktiskt möjligt i Bash. Det kallas kommandosubstitution (eng: command substitution) och kan åstadkommas på två vis. Dels med syntaxen **`$(kommando)`**. Dels med syntaxen **`` `kommando` ``** . I det senare fallet kallas fnuttarna för *backticks* på engelska. Det finns säkert något namn på svenska för dem också men backtick är ett etablerat namn.

Substitution fungerar som vid variabler och *brace expansion*, att Bash evaluerar (ung: utvärderar eller räknar ut) uttrycket innan det används av nästa kommando. Här är några enkla exempel som visar båda syntaxerna:

```
$ echo "Klockan är $(date +%T) och det är $(date +%A) idag."
Klockan är 08:33:22 och det är fredag idag
$ echo "Klockan är `date +%T` och det är `date +%A` idag."
Klockan är 08:34:20 och det är fredag idag.
```

Det är viktigt att förstå att det är Bash som expanderar variabler, krullparenteser och kommandosubstitution innan resultatet används av till exempel `echo`.

Uppgift:

Leta upp filen a_few_urls.txt från tidigare övningar. Skriv en echo-sats som resulterar i följande textutskrift (med hjälp av kommandosubstitution):
`Den första URLen är http://www.gu.se/bazinga`

Där URLen kommer från första raden i filen.

Tips: Prova först att med hjälp av `head` skriva ut enbart första raden. När det fungerar, så vet du vad du ska använda för kommando i echo-satsens kommandosubstitution.

Räkna med problem

Man kan faktiskt använda Bash för att lösa enkla matematiska aritmetiska räkneproblem genom ytterligare en sorts expansion. Genom att använda syntaxen `$((uttryck))` där

uttrycket är till exempel `11 * 9` så kan man få Bash att räkna ut resultatet, vilket kan användas som utskrift eller som argument till andra program och så vidare. Exempel:

```
$ echo "Det är $(( 60 * 60 * 24 )) sekunder på ett dygn"
Det är 86400 sekunder på ett dygn
```

Eller kanske ännu snyggare:

```
$ SECS_PER_MIN=60; MINS_PER_HOUR=60; HOURS_PER_DAY=24
$ echo "Det är $(( SECS_PER_MIN * MINS_PER_HOUR * HOURS_PER_DAY )) sekunder på ett dygn"
Det är 86400 sekunder på ett dygn
```

Notera att man inte behöver skriva dollartecken före variabelnamnen inuti `$(( var ))`.

Självklart skulle man kunna använda kommandot `bc` (som är en kalkylator) med kommandosubstitution i stället med hjälp av en pipeline:

```
$ echo "Det är $(echo "$SECS_PER_MIN * $MINS_PER_HOUR * $HOURS_PER_DAY" | bc) sekunder på ett
dygn"
Det är 86400 sekunder på ett dygn
```

(Raden ovan fick inte rum på en rad här i häftet).

Följande är några av de aritmetiska operationer är möjliga med syntaxen $((expr)):

- `+ -` Addition, subtraktion
- `* / %` Multiplikation, (heltals)division, rest vid heltalsdivision
- `**` Exponering (“upphöjt till”)

Exempel:

```
$ echo "En byte kan representera $(( 2**8 )) olika värden."
En byte kan representera 256 olika värden.
$ echo "5 ben till 3 hundar blir $((5/3)) ben per hund"
5 ben till 3 hundar blir 1 ben per hund
$ echo "Det blir $((5%3)) ben över."
Det blir 2 ben över.

$ echo "16 bitar kan lagra $(( 2**16 - 2**8 )) fler värden än en byte."
16 bitar kan lagra 65280 fler värden än en byte.
```

En användbar finess med `$(( expr ))` är att vi kan manipulera variabler i uttrycket. Särskilt variabler med numeriska (heltal) värden är det vanligt att man till exempel ökar med ett eller minskar med ett.

För att öka en variabel med heltalsvärde med ett kan vi använda följande syntax (där man inte får sätta $ före variabelnamnet och inte före parenteserna om vi inte vill använda värdet på variabeln):

```
(( counter++ ))
```

För att minska med ett är syntaxen:

```
(( counter-- ))
```

Exempel:

```
$ counter=0
```

```
$ echo "counter är nu $counter"; ((counter++))
counter är nu 0
$ echo "counter är nu $counter"; ((counter++))
counter är nu 1
$ echo "counter är nu $counter"; ((counter++))
counter är nu 2
$ echo "counter är nu $counter"
counter är nu 3
```

Uppgifter:
Skriv en echo-sats som använder `$(( uttryck ))` för att skriva ut hur många timmar det går på en vecka, en månad (med 30 dagar), ett år (med 365 dagar).

Skriv en echo-sats som använder `$(( uttryck ))` för att skriva ut hur många minuter 3745 sekunder är och hur många sekunder "som blir över".

Skrive en echo-sats som verifierar att utskriften är korrekt (räkna baklänges från ditt resultat).

Vissa av uppgifterna har lösningsförslag i slutet på häftet.

Loopar - for, while - Att göra något upprepade gånger

Vi sade tidigare att man kan använda dubbla parenteser utan dollartecken för att öka eller minska värdet på en heltalsvariabel med ett. Man kan göra fler saker inom dubbla parenteser om man sätter semikolon emellan.

For-loopens första led har följande delar:
`(( initialt-värde; kontroll-om-fortsätta; förändring))` .
Till exempel: `(( i = 0; i < 10; i++ ))`

Initialt sätts en variabel, t ex `i`, till noll. Sedan kontrolleras om `i` är mindre än 10, i vilket fall vi kommer exekvera kommandona mellan `do` och `done` (som är nästa led i for-loopen).

Här är ett exempel:
```
$ for ((i = 0; i < 10; i++))
> do
> echo "Variabeln i är nu $i."
> done
Variabeln i är nu 0.
Variabeln i är nu 1.
Variabeln i är nu 2.
Variabeln i är nu 3.
Variabeln i är nu 4.
Variabeln i är nu 5.
Variabeln i är nu 6.
Variabeln i är nu 7.
Variabeln i är nu 8.
```

```
Variabeln i är nu 9.
```

Lägg märke till att vi fick en sekundär-prompt: > eftersom vi bröt rad efte första ledet. Bash väntar tålmodigt på att vi ska skriva klart loopen med `for`-ledet, och `do`-`done`-ledet.

Om vi inte bryr oss om radbrytningar, så kan vi skriva allt på en rad med semikolon mellan leden:

```
$ for ((i = 0; i < 10; i++)); do echo "Variabeln i är nu $i."; done
Variabeln i är nu 0.
Variabeln i är nu 1.
Variabeln i är nu 2.
Variabeln i är nu 3.
Variabeln i är nu 4.
Variabeln i är nu 5.
Variabeln i är nu 6.
Variabeln i är nu 7.
Variabeln i är nu 8.
Variabeln i är nu 9.
```

Uppgifter:
Skriv en for-loop som skriver ut talen 0 till 19 på varsin rad.

Skriv en for-loop som räknar ut summan av talen 0-10 . Skriv ut värdet på variabeln efter done (efter loopen).
Tips: Använd en variabel som sätts till 0 före loopen. I do-done-blocket, sätt varabeln till sitt nuvarande värde plus loop-variabeln i.
Tips: För att påverka en variabel med hjälp av dess nuvarande värde kan man skriva:

```
$ my_var=$((my_var + 1))    # nuvarande värde plus ett
```

Om du gjort rätt bör du få summan 55. “Bevis”:

```
$ echo {0..10}
0 1 2 3 4 5 6 7 8 9 10
$ echo {0..10} | tr ' ' '+'
0+1+2+3+4+5+6+7+8+9+10
$ echo {0..10} | tr ' ' '+' | bc
55
```

Du kan snabbt räkna ut i huvudet att det stämmer också om du betänker att serien 0-10 innehåller följande lämpliga par av tal:

- 0 + 10
- 1 + 9
- 2 + 8
- 3 + 7
- 4 + 6
- 5

Nu ska vi öva lite på while-loopen.

While-loopen har följande form:
`while ((kontroll-av-variabel)); do kommandon; done`

Två saker att tänka på: Det är lämpligt att skapa loop-variabeln innan while-loopen, eftersom loopen endast har en del i “huvudet”, det vill säga kontroll om `do-done`-blockets kommandon ska utföras eller inte. Det är också lämpligt att förändra loop-variabeln i blocket, så att kontrollen så småningom leder till att loopen avbryts.

Det är nämligen så att kontrollen avgör om loopen ska fortsätta eller inte (precis som mittendelen av for-loopens huvud, till exempel `i < 10`). Om du glömmer att förändra i, så kommer i alltid vara mindre än 10, exempelvis (om du börjar med i=0) och då kommer do-done utföras oändligt många gånger (eller tills du avbryter med Ctrl-C).

Kort exempel, så du ser hur strukturen ser ut:

```
$ while (( i < 10 ))
> do echo "Variabeln i är nu $i"
> ((i++))
done
Variabeln i är nu 0
Variabeln i är nu 1
Variabeln i är nu 2
Variabeln i är nu 3
Variabeln i är nu 4
Variabeln i är nu 5
Variabeln i är nu 6
Variabeln i är nu 7
Variabeln i är nu 8
Variabeln i är nu 9

$ while (( i < 10 )); do echo "Variabeln i är nu $i"; ((i++));done
Variabeln i är nu 0
Variabeln i är nu 1
Variabeln i är nu 2
Variabeln i är nu 3
Variabeln i är nu 4
Variabeln i är nu 5
Variabeln i är nu 6
Variabeln i är nu 7
Variabeln i är nu 8
Variabeln i är nu 9
```

Uppgifter: Räkna ut summan av talen 0-100 med en while-loop. Glöm inte att påverka loopvariabeln i do-done-blocket (förslagsvis efter du har uppdaterat summa-variabeln). Glöm inte att deklarera och sätta variabeln i (som är det vanligaste namnet på loop-variabeln) till noll innan while-loopen och glöm inte heller att “nollställa” summa-variabeln innan while-loopen (den kanske fortfarande har värdet 55 från förra övningen).

En användbar variant av for-loopen använder inte dubbelparenteser, utan en lista med strängar i stället:

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

Det som gör denna variant så användbar (och vanlig) är att man kan använda kommandosubstitution för att skapa listan:

```
$ for file in $(ls *.txt); do echo "$file"; done
a_few_urls.txt
urls.txt
$
```

Tänk dock på att exemplet ovan inte fungerar väl med filer eller kataloger med mellanslag i namnen. Vilket är en anledning till att alltid undvika mellanslag som en del av fil- eller katalognamn. Men det är viktigt att tänka på att en sådan fil i kombination med loopen kommer betrakta varje del av namnet som flera strängar. Filen “mitt dåliga filnamn.txt” kommer betraktas som tre strängar i loopen: “mitt”, “dåliga” och “filnamn.txt” . Nu har vi varnat dig, i vart fall.

Uppgifter:
Använd `for` med kommandosubstitution av kommandot `grep` för att enbart skriva ut URLer med domännamn som börjar på `www.` i loopen (från a_few_urls.txt).

Gör samma sak men skriv bara ut URLer som *inte* har domännamn som börjar på `www.` .

För säkerhets skull bör du säkerställa att `www` är början på domännamnet (kommer efter `//`) och att det följs av en verklig punkt (i `grep` betyder ju `.` “vilket tecken som helst” medan `\.` betyder “en faktisk punkt”).

Tips: Testa först ditt `grep`-kommando direkt i kommandoraden innan du använder den för att producera listan till `for`-loopen. På så vis så vet du att det kommer fungera i kommandosubstitutionen.

OBS! Detta är blott en konstruerad övning för att träna på `for`-loopen. Det räcker ju att använda grep direkt. Lägg till en rubrik i utskriften av loopen, så ser du att det inte räcker med `grep`!

Skapa arkiv av filer - zip, tar med mera

För att komprimera en fil kan du använda kommandot zip:

```
$ zip urls.txt.zip urls.txt
  adding: urls.txt (deflated 73%)
$ ls -l urls.txt*
-rw-rw-r-- 1 rikard rikard 15278 aug 21 16:23 urls.txt
-rw-rw-r-- 1 rikard rikard  4239 aug 23 10:31 urls.txt.zip
$ ls -lh urls.txt*
-rw-rw-r-- 1 rikard rikard  15K aug 21 16:23 urls.txt
-rw-rw-r-- 1 rikard rikard 4,2K aug 23 10:31 urls.txt.zip
```

Som du såg, så är syntaxen: `zip zip-fil-att-skapa fil(er)-att-komprimera .`

```
$ man zip | grep -i -A2 synopsis
SYNOPSIS
       zip  [-aABcdDeEfFghjklLmoqrRSTuvVwXyz!@$]  [--longoption
...]  [-b path] [-n suffixes] [-t date]
       [-tt date] [zipfile [file ...]]  [-xi list]
$
```

Den enkla syntaxen av ovanstående som vi använder i vårt exempel är:
`zip zipfile file ...`
där `zipfile` är namnet på den komprimerade filen vi vill skapa och `file ...` är den eller de filer vi vill komprimera.

Det finns en användning till som vi vill att du ska känna till. Det är att man kan skapa en zipfil av ett helt katalogträd. Det är ju smidigt om du ska skicka en hel katalog med allt dess innehåll till en annan dator eller en kollega (eller som inlämningsuppgift till din lärare).

```
$ tree
.
├── notes.txt
└── övningar
    ├── a_few_urls.txt
    ├── fixed2.csv
    ├── fixed.csv
    ├── test
    │   └── some-words.txt
    ├── tig015h19.csv
    └── urls.txt

2 directories, 7 files
```

Låt oss skapa en zip fil med hela katalogen övningar. Syntaxen är `zip -r zipfil katalog .`

Exempel:

```
$ zip -r exercises.zip övningar/
  adding: övningar/ (stored 0%)
```

```
  adding: övningar/test/ (stored 0%)
  adding: övningar/test/some-words.txt (deflated 15%)
  adding: övningar/urls.txt (deflated 73%)
  adding: övningar/fixed.csv (deflated 90%)
  adding: övningar/tig015h19.csv (deflated 89%)
  adding: övningar/a_few_urls.txt (deflated 36%)
  adding: övningar/fixed2.csv (deflated 90%)
$ du -sh övningar/    # du -sh prints the size of a directory
68K   övningar/
$ ls -lh exercises.zip
-rw-rw-r-- 1 rikard rikard 9,1K aug 23 10:40 exercises.zip
$
```

För att packa upp en zip-fil, använd `unzip zip-fil` .

```
$ mkdir tmp
$ mv exercises.zip tmp/
$ cd tmp/
$ unzip exercises.zip
Archive:  exercises.zip
   creating: övningar/
   creating: övningar/test/
  inflating: övningar/test/some-words.txt
  inflating: övningar/urls.txt
  inflating: övningar/fixed.csv
  inflating: övningar/tig015h19.csv
  inflating: övningar/a_few_urls.txt
  inflating: övningar/fixed2.csv
$ ls
exercises.zip  övningar
```

Notera att exercises.zip finns kvar. Det är bara kopior som skapas när man packar upp filen.

Uppgifter:
Skapa zip-filer med komprimerade kopior av några av dina kataloger (välj kataloger med många filer i för störst effekt). Kontrollera med `du -sh` hur stora katalogerna är och kontrollera med `ls -lh` hur stora zip-filerna blev. Notera att små filer faktiskt kan bli större när man zippar dem (zip-formatet har lite overhead och “lönar sig” först på filer av en viss storlek).

Prova nu att packa upp någon eller några av filerna i en nyskapad katalog. Se ovan för syntax-exempel.

Kommandona tar och gzip

Det finns två (eller egentligen tre) kommandon till (som ofta används i kombination dessutom) som vi vill att du känner till. Det första är `tar` (från eng: tape archive).

Kommandot tar hanterar arkiv av filer och kataloger (utan komprimering). Syntaxen är lite annorlunda:

- `tar cf arkivfil.tar files-or-dirs` — Skapa filen arkivfil.tar från argumenten
- `tar xf arkivfil.tar` — Packa upp arkivfil.tar i aktuell katalog (arkivfil.tar finns kvar efteråt)
- `tar tf arkivfil.tar` — Lista innehållet i arkivfil.tar

Det andra (och tredje) kommandot är `gzip` (och kompisen `gunzip`). Det är ett kommando för att komprimera och packa upp filer (som kan vara hela tar-arkiv).

```
$ man gzip | grep -iA3 synopsis
SYNOPSIS
       gzip [ -acdfhklLnNrtvV19 ] [--rsyncable] [-S suffix] [ name
...  ]
       gunzip [ -acfhklLnNrtvV ] [-S suffix] [ name ...  ]
       zcat [ -fhLV ] [ name ...  ]
```

För att komprimera maximalt, använd flaggan `-9` . En skilnad från zip/unzip är att gzip komprimerar en fil så att originalet försvinner och nu “ligger i gz-filen”:

```
$ gzip -9 urls.txt
$ ls
a_few_urls.txt  fixed2.csv  fixed.csv  test  tig015h19.csv
urls.txt.gz
$ # urls-txt är borta och finns nu komprimerad i urls.txt.gz
```

När man packar upp filen igen, med i exemplet ovan `gunzip urls.txt.gz`, så “försvinner” gz-filen och originalfilen skapas i stället:

```
$ ls -lh urls.txt.gz
-rw-rw-r-- 1 rikard rikard 4,0K aug 21 16:23 urls.txt.gz
$ gunzip urls.txt.gz
$ ls
a_few_urls.txt  fixed2.csv  fixed.csv  test  tig015h19.csv  urls.txt
$ ls -lh urls.txt
-rw-rw-r-- 1 rikard rikard 15K aug 21 16:23 urls.txt
$
```

Uppgifter:
Använd gzip och gunzip för att komprimera och packa upp olika textfiler. Notera storlek före och efter komprimering.

archive.tgz eller archive.tar.gz

Det är så vanligt att man först skapar en tar-fil av ett filträd (en katalog med allt därunder) och sedan använder gzip för att komprimera tar-filen, att det till och med har ett speciellt

namn: tar-boll (eng: tar ball). Du kan använda suffixet .tgz eller .tar.gz om du vill följa standarden i Unix-världen.

Så här går det till att skapa:

```
$ tree
.
└── övningar
    ├── a_few_urls.txt
    ├── fixed2.csv
    ├── fixed.csv
    ├── test
    │   └── some-words.txt
    ├── tig015h19.csv
    └── urls.txt

2 directories, 6 files
$ tar cf exercises.tar övningar/
$ gzip -9 exercises.tar
$ ls
exercises.tar.gz  övningar
$ du -sh övningar/
68K   övningar/
$ ls -lh exercises.tar.gz
-rw-rw-r-- 1 rikard rikard 6,2K aug 23 11:04 exercises.tar.gz
$
```

Detta är så vanligt att man faktiskt kan använda tar för att göra båda stegen (även packa upp):

```
$ tree
.
└── övningar
    ├── a_few_urls.txt
    ├── fixed2.csv
    ├── fixed.csv
    ├── test
    │   └── some-words.txt
    ├── tig015h19.csv
    └── urls.txt

2 directories, 6 files
$ tar czf exercises.tgz övningar/
$ ls -lh exercises.tgz
-rw-rw-r-- 1 rikard rikard 6,5K aug 23 11:06 exercises.tgz
$ rm -rf övningar/
$ ls
exercises.tgz
```

```
$ tar xvf exercises.tgz
övningar/
övningar/test/
övningar/test/some-words.txt
övningar/urls.txt
övningar/fixed.csv
övningar/tig015h19.csv
övningar/a_few_urls.txt
övningar/fixed2.csv
$ ls
exercises.tgz  övningar
```

Flaggorna till tar är:

- c create
- f with file name
- v be verbose about it (skriv ut vad du gör när du gör det)
- x extract (packa upp)
- z zipped (komprimera/packa upp)

Så: `tar cvzf tarball.tgz directory` skapar filen `tarball.tgz` med ett zippat tar-arkiv av `directory`.

Uppgifter:
Använd tar och gz för att skapa en tarboll och packa upp den i en nyskapad katalog någonstans. Notera storlek före och efter.

Gör samma sak med enbart `tar` (använd flaggorna `czf` för att “create zipped file” respektive `xzf` för att “extract zipped file”).

Det finns inga lösningsförslag för denna del. Det är meningen att du ska öva tills det fungerar och fråga någon om hjälp om det inte fungerar (lärare, kollega, internet, handledare osv).

Lösningsförslag på vissa av uppgifterna

Arbeta med textfiler

Vilka undervisningsformer är vanligast?

Vårt exempel på hur man får fram vilka undervisningsformer som är vanligast:

```
$ cut -d ',' -f10 < tig015h19.csv | grep -v Undervisningstyp | sort | uniq -c | sort -rnk1
     20 Handledning
     16 Workshop
     13 Föreläsning
      4 Lärare/student möte
      4 "Föreläsning Projektarbete"
      1 Introduktion
      1 Digital tentamen
```

Steg-för-steg:

```
cut -d ',' -f10 < tig015h19.csv
# skriv ut kolumn 10, (använd komma som avskiljare) från filen

| grep -v Undervisningstyp
# från dessa rader, bortse från rader som innehåller
# Undervisningstyp

| sort
# från dessa rader, sortera dem

| uniq -c
# från dessa rader, ta bort dubbletter och räkna hur många
# dubbletter det var

| sort -rnk1
# från dessa rader, sortera dem fallande med avseende på kolumn 1,
# numeriskt
```

Manipulera URLer

Vår förslag på lösning på att klippa bort protokollen från URLerna:

```
$ cut -d '/' -f3- < a_few_urls.txt
www.gu.se/bazinga
ait.gu.se/forskning/journalister
www.elsewhere.org/pomo/
snarxiv.org/vs-arxiv/
www.physics.nyu.edu/faculty/sokal/afterword_v1a/afterword_v1a_single
file.html
```

Vårt lösningsförslag på att filtrera ut endast domännamn:

```
$ cut -d '/' -f3 < a_few_urls.txt
www.gu.se
ait.gu.se
www.elsewhere.org
snarxiv.org
www.physics.nyu.edu
```

Vårt lösningsförslag på att hitta URLer med en fil med punkt i filnamnet i slutet av URLen:

```
$ grep '.*\.[a-z]*$' a_few_urls.txt
http://www.physics.nyu.edu/faculty/sokal/afterword_v1a/afterword_v1a_singlefile.html
```

Förklaring:
Vi vill hitta sista delen av URL:en, så vi använder mönstret:

```
'.*\.[a-z]*$'
```

Det vill säga:
noll eller flera tecken: **`.*`** följt av
en punkt: **`\.`** följt av
noll eller flera bokstäver: **`[a-z]`** följt av
slut på raden (inget mer): **`$`**

Med andra ord, raden ska sluta med en punkt och endast bokstäver efter punkten.

Vårt lösningsförslag för att gruppera domänerna med avseende på toppdomän:

```
$ cat a_few_urls.txt  | cut -d '/' -f3 | rev | sort | rev

ait.gu.se
www.gu.se
www.elsewhere.org
snarxiv.org
www.physics.nyu.edu
```

Vi tar ett exempel med de femtio första domänerna från den stora filen urls.txt också, så ni ser att det fungerar generellt:

```
$ head -50 urls.txt | cut -d '/' -f3 | rev | sort | rev
www.presidencia.gob.pa
www.diwan.gov.qa
www.presidencia.gob.ec
www.corteconstitucional.gob.ec
www.nc.ee
www.supremecourt.ge
www.government.gov.ge
www.pcm.gob.pe
www.ernestkoroma.org
www.kktcb.org
www.enenkio.org
www.ttlawcourts.org
www.somaligovernment.org
www.ipu.org
www.pmo.gov.sg
www.cabinet.gov.sg
www.camnet.com.kh
www.tpk.fi
www.vicepresidencia.gob.ni
www.cook-islands.gov.ck
www.priu.gov.lk
www.nio.gov.uk
www.mash.gov.al
www.statehouse.gm
www.boliviaweb.com
www.pmis.gov.mn
www.angola.gov.ao
www.domstol.no
www.assembleenationale.fr
www.vlada.hr
www.tsk.tr
www.althingi.is
www.palauoek.net
www.palazzochigi.it
www.aph.gov.au
ec.europa.eu
www.meh.hu
www.kormany.hu.
www.president.lv
www.fbi.gov
www.csj.gob.sv
www.gov.rw
www.presidencia.gob.mx
www.scjn.gob.mx
www.diputados.gob.mx
www.government.by
www.gov.ky
www.pmo.gov.my
www.vlada.cz
www.mzcr.cz
```

Som en rolig sak, kan vi bjuda på en lösning där vi faktiskt kan sortera med avseende på toppdomän också. Problemet är att veta vilket fält vi ska sortera på, toppdomänen kommer ju på olika plats beroende på vad som kommer före, t ex i:

```
www.nc.ee        ## ee har plats 3, med punkt som avskiljare
www.pmo.gov.sg   ## sg har plats 4, med punkt som avskiljare
```

Strategin man kan använda sig av är att använda ett program för att skapa rader med toppdomänen kopierad till början av raden, sortera och sedan klippa bort toppdomänen igen:

```
www.nc.ee      -> ee.www.nc.ee      -> www.nc.ee
www.pmo.gov.sg -> sg.www.pmo.gov.sg -> www.pmo.gov.sg
```

Man kan använda awk eller sed, till exempel, för att ta sista delen (toppdomänen) och skriva ut den före originalraden. Vi lär inte ut sed och awk i detta material, dock, det är för stort. Men vi kan visa lösningsförslag för dem som är intresserade:

```
$ cat a_few_urls.txt  | cut -d '/' -f3 |
     sed -e 's/\(.*\)\.\([a-z]*$\)/\2.\1.\2/g' |
     sort |cut -d '.' -f2-

www.physics.nyu.edu
snarxiv.org
www.elsewhere.org
ait.gu.se
www.gu.se
```

Vi tar 15 rader från stora filen för att visa att det fungerar:

```
$ head -15 urls.txt  | cut -d '/' -f3 | sed -e 's/\(.*\)\.\([a-z]*$\)/\2.\1.\2/g'
lk.www.priu.gov.lk
my.www.pmo.gov.my
au.www.aph.gov.au
fr.www.assembleenationale.fr
ge.www.supremecourt.ge
gov.www.fbi.gov
by.www.government.by
org.www.somaligovernment.org
net.www.palauoek.net
kh.www.camnet.com.kh
com.www.boliviaweb.com
ck.www.cook-islands.gov.ck
.www.kormany.hu.
al.www.mash.gov.al
ky.www.gov.ky
```

Samma sak med awk:

```
$ head -15 urls.txt | cut -d '/' -f3 | awk -F '.' '{print $NF,$0}'|sort | cut -d ' ' -f2-
www.mash.gov.al
www.aph.gov.au
www.government.by
www.cook-islands.gov.ck
www.boliviaweb.com
www.assembleenationale.fr
www.supremecourt.ge
www.fbi.gov
www.kormany.hu
www.camnet.com.kh
www.gov.ky
www.priu.gov.lk
www.pmo.gov.my
www.palauoek.net
www.somaligovernment.org
```

Det är upp till er (som är intresserade) att försöka lista ut hur sed- och awk-raderna fungerar. Principen är alltså att kopiera in det sista fältet först i raden, sortera, ta bort det från först i raden igen.

Att redigera kommandoraden samt en del tricks

Lista filer med glob-uttryck

Till exempel:

```
$ ls [au]*      # lista alla filer som börjar på a eller u
$ ls [au]*.txt  # som ovan men också slutar på txt
$ ls [^au]*     # lista alla filer som inte börjar på a eller u
```

Använd brace expansion { }

Till exempel:

```
$ echo ap{skaft,anage,-fonden,elsin,pell,a,parat}
apskaft apanage ap-fonden apelsin appell apa apparat
```

Till exempel:

```
$ echo grupp{1..20}
grupp1 grupp2 grupp3 grupp4 grupp5 grupp6 grupp7 grupp8 grupp9
grupp10 grupp11 grupp12 grupp13 grupp14 grupp15 grupp16 grupp17
grupp18 grupp19 grupp20
```

Till exempel:

```
$ echo grupp{20..5..5}      # från 20 till 5, fem steg i taget
grupp20 grupp15 grupp10 grupp5
```

Variabler

Lösningsförslag:

```
$ echo "Det är $SECS_PER_MIN sekunder på en minut, $MINS_PER_HOUR
minuter på en timme och $HOURS_PER_DAY timmar per dygn."
Det är 60 sekunder på en minut, 60 minuter på en timme och 24 timmar
per dygn.
```

Kommandosubstitution

Lösningsförslag:

```
$ echo "Den första URLen är $(head -1 a_few_urls.txt)"Den första
URLen är http://www.gu.se/bazinga
```

Räkna med problem

Lösningsförslag:

```
$ echo "Det går $(( 24 * 7 )) timmar på en vecka, $(( 24 * 30 ))
timmar på en månad och $(( 60 * 24 * 365)) timmar på ett år."
Det går 168 timmar på en vecka, 720 timmar på en månad och 525600
timmar på ett år.

$ echo "3745 sekunder är $((3745/60)) minuter och $((3745%60))
sekunder."
3745 sekunder är 62 minuter och 25 sekunder.

$ echo "62 minuter och 25 sekunder är $((62 * 60 + 25)) sekunder."
62 minuter och 25 sekunder är 3745 sekunder.
```

Loopar

Lösningsförslag:

Skriv ut talen 0-19 på varsin rad:

```
$ for ((i = 0; i < 20; i++)); do echo "Variabeln i är nu $i."; done
```

Använd en for-loop för att räkna ut summan av talen från 0 till 10:

```
$ sum=0
$ for ((i = 0; i < 11; i++))
> do sum=$((sum + i))
> done
$ echo "Talen 0-10 har summan $sum."
Talen 0-10 har summan 55.
```

Använd en while-loop för att räkna ut summan av talen mellan 0 och 100:

```
$ i=0
$ sum=0
$ while (( i < 101 ))
>  do
>  sum=$((sum + i))
>  ((i++))
> done
$ echo "Summan av talen 0-100 är $sum"
Summan av talen 0-100 är 5050
$ # eller:
$ sum=0; i=0
$ while (( i < 101 ));  do  sum=$((sum + i));  ((i++)); done
$ echo "Summan av talen 0-100 är $sum"
Summan av talen 0-100 är 5050
```

Lösningsförslag på `for variabel in $(kommando)`:

```
$ for url in `grep '//www\.' a_few_urls.txt`; do echo "$url"; done
http://www.gu.se/bazinga
http://www.elsewhere.org/pomo/
http://www.physics.nyu.edu/faculty/sokal/afterword_v1a/afterword_v1a
_singlefile.html
$ # eller:
$ for url in $(grep '//www\.' a_few_urls.txt); do echo "$url"; done
http://www.gu.se/bazinga
http://www.elsewhere.org/pomo/
http://www.physics.nyu.edu/faculty/sokal/afterword_v1a/afterword_v1a
_singlefile.html
$
$ # “motsatsen”, rader utan www.:
$ for url in $(grep -v '//www\.' a_few_urls.txt); do echo "$url";
done
https://ait.gu.se/forskning/journalister
http://snarxiv.org/vs-arxiv/
$
```

Slutord

Det finns fler övningar, föreläsningsfilmer och exempel på vår wiki (länk nedan).

Hör gärna av er om ni hittar fel i språk eller sak, så hjälps vi åt att förbättra materialet. Om ni hittar fel och vill vara med under “tack till...” i kommande utgåvor, så berätta detta när ni hör av er. Det går naturligtvis bra att höra av sig anonymt eller utan att vilja ha med sitt namn också.

Om du som elev, student eller privatperson gillade materialet och vill lära dig mer om datorer och programmering kanske något av det kursmaterial vi skrivit passar dig. Vi har material (teori, övningar, lösningsförslag och videofilmer) för bland annat Java, Bash och Databaser och du hittar detta enklast om du går till:

> http://wiki.juneday.se

Om du jobbar med undervisning och vill få rätt att använda vårt material i dina kurser är det bara att höra av sig till oss på info@juneday.se. Vi har lärarinstruktioner, lösningsförslag, rättningsscript med mera som stöd för läraren och handledare.

Våren 2019
Henrik Sandklef och Rikard Fröberg

Slut

Kontakt: info@juneday.se

Skicka felrapporter och klagomål eller annat till:
rikard.froberg@ait.gu.se
henrik.sandklef@ait.gu.se

Vill du använda vårt material i din undervisning? Kontakta författarna.