Kursinfo

Som framgått av tidningen kommer vi att hålla en kurs i Unix för max 30 intresserade personer. Vi kommer att börja vecka 13 och köra 5 veckor framöver med uppehåll under omtentaperioden på LTH.

Varje vecka blir det 4 timmars undervisning med en 2 timmars föreläsning för alla och en 2 timmars laboration i grupper om 6 st. Totalt omfattar kursen alltså 20 timmar.

Kurslitteraturen kommer att bestå dels av en utmärkt bok:
"The Unix Programming Environment" av Kernighan & Pike (pris i bokhandeln ca 350:-), dels några hundra sidor kopior.

För kursen inklusive kurslitteraturen kommer vi att ta ut en avgift på 300:- per person. Eftersom vi kommer att driva kursen i samarbete med ett studieförbund kommer vi att få lite bidrag, varför den som fullföljer kursen kommer att få 50:- i retur.

Antalet platser är som sagt begränsat och teckningslistor finns uppsatta på en anslagstavla i den korridor där ETF har sina lokaler.

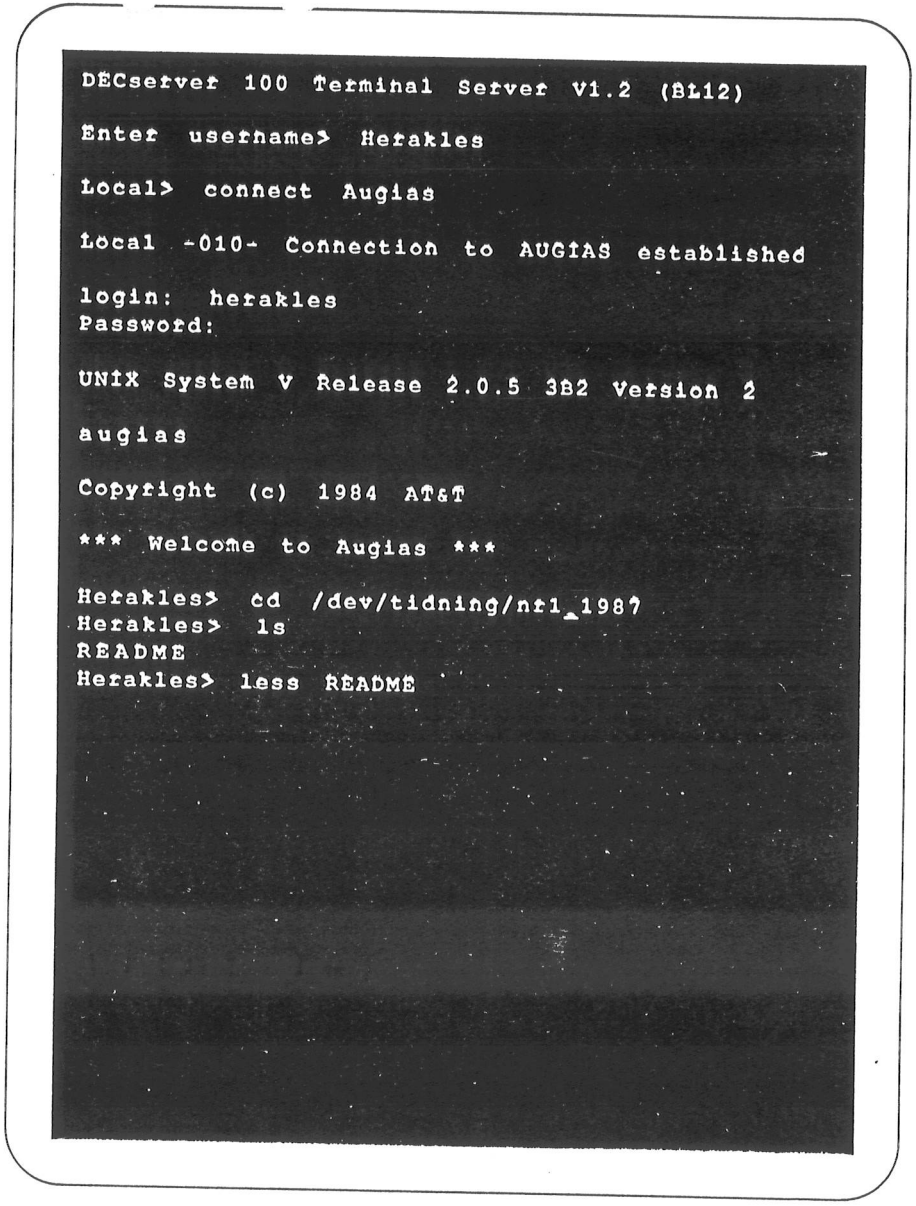
Förkunskapskrav: För att man ska kunna tillgodogöra sig kursen bör man ha grundläggande kunskaper om någon dator (exempelvis VAX) och dess operativsystem (i så fall VMS). Dessutom bör man ha grundläggande kunskaper om minst ett programspråk, lämpligen Pascal.

Preliminär kursplan:

1. Introduktion till Unix
2. Mer om Unix
3. Introduktion till programspråket C
4. Mer om C
5. Övrigt om Unix

Att vi lär ut programspråket C beror på att det är det enda programmeringsspråk man kan vara säker på att finna på en Unix-maskin. Det är på många sätt mycket likt Pascal och vi kommer inte att lära ut alla dess finesser, utan i stort sett lära er skriva samma program ni idag kan skriva i Pascal i C istället. Om man liknar Pascal vid en bil kan man likna C vid samma bil, fast med borttagna säkerhetsbälten och trimmad motor.

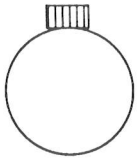
Välkomna!



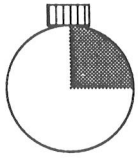
LTH:S DATORFÖRENING

Om en minut!

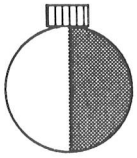
Det tar dig en minut att läsa den här sidan. Gör det! I så fall vet du om resten av den här tidningen är något för dig. Om en minut alltså.



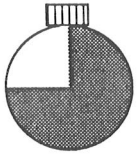
Först kommer vi att berätta om att LTH:s Datorförening fått en egen dator. Det är en minidator i VAX-klassen men med ett helt annat operativsystem - Unix. Den heter Augias och går i dagsläget att nå från terminalrummen NEPTUNUS, PLUTO, ALFA och BETA.



Fast nu har jag narrats lite - datorn är inte vår på riktigt. Det är Professor Lars Philipson vid institutionen för DatorSystem teknik som fått den som donation av Olivetti. Men han tyckte att den bäst kunde användas för att lära eleverna här vid skolan ett och annat om operativsystemet Unix och bad oss sköta om den. Vi vill därför passa på och rikta ett varmt **TACK** till Lars Philipson.



När vi berättat om datorn fortsätter vi med operativsystemet. Först en kort artikel med titeln "The History of Unix". När ni läst den inser ni att ett operativsystem, som tillkommit därför att dess skapare ville kunna köra ett spelprogram, inte kan vara tråkigt att lära sig. Då hugger ni säkert tänderna i artikeln "Kort introduktion till Unix".



Nu kan det vara nog med teori för en dag. På slutet berättar vi lite om LTH:s Datorförening och visar några tjusiga bilder på oss som sitter i styrelsen. Det är för att du ska veta vilka vi är, så du vet vilka du ska fråga om du vill veta mer.

Nu har visst minuten gått, men jag har sparat det bästa till sist, så ge mig 15 sekunder till. Allra sist gör vi nämligen reklam för en kurs i Unix vi tänker starta om en vecka. Det finns bara 30 platser, så först till kvarn...

Robert Haneklou
Ordförande

Augias (realdel)

Augias är en dator av typen AT&T 3B2/400. Den har skänkts av företaget Olivetti som är säljer AT&T:s datorer i Europa. Professor Lars Philipson har haft vänligheten att låta oss i LTH:s Datorförening ta hand om den.

Eftersom dess operativsystem - Unix - ännu så länge är i det närmaste okänt (bland eleverna) här på skolan tänkte vi börja vår verksamhet med att ge 30 personer chansen att gå en kurs i Unix (vill du bli en av de 30? -Se baksidan av tidningen). När det på så sätt finns 30 personer med grundläggande kunskaper i Unix som man kan fråga om något blir fel, tänker vi låta även övriga medlemmar få tillgång till Augias. Exakt när och hur detta kommer att ske ber vi att få återkomma om. I dagsläget tar vi oss dock friheten att endast ge de som går kursen tillgång till Augias.

Varför är då Augias intressant? Ja, dels har den ett intressant operativsystem - Unix. Förutom de fördelar med Unix som framgår av andra artiklar i den här tidningen, så finns det faktiskt ett 20-tal andra datorer här i huset som kör Unix. Ännu her de inte kommit in i grundutbildningen, men om du har mer än något enstaka år kvar på här på skolan tror vi chansen är stor att du kommer att komma i kontakt med Unix i någon kurs innan du slutar. Unix kan alltså förutom att det är intressant vara bra att kunna.

Eftersom Augias inte är avsedd att användas i vanlig undervisning kommer vi att ha betydligt friare regler än övriga datorer i huset när det gäller vad som är tillåtet och inte. En regel kommer vi dock att hålla benhårt på: **Man får inte förstöra för andra.**

För den tekniskt intresserade kommer här lite data om Augias. Du som inte är så intresserad av torra fakta hoppar lämpligen till nästa sida där mer spirituell läsning erbjudes.

Augias har: 32-bitars WE 32100 mikroprocessor med 10 MHz klockfrekvens

Primärminne - 2 MB RAM idag, med en ett löfte om utbyggnad till 4 MB så småningom.

Sekundärminne - 2 * 72 Mb Winchester minnen. 1 st 5.25 tums skivminne för överföring av programvara mm. En diskett rymmer 720 Kb. 1 st 5.25 tums "cartridge" bandstation för back-uper. En "cartride" rymmer 23 Mb

Diverse kortplatser för utbyggnad. Olivetti har ställt en utbyggnad med ethernet-anslutning i sikte.

2 parallell-portar och 10 serie-portar. 4 serieportar är i dag anslutna till E-husets terminalservernät. 1 serieport är ansluten till PERDIX för uucp-kommunikation. 1 serieport är via PERDIX ansluten till skivaren i NEPTUNUS.

Operativsystemet är i dagsläget Unix system V version 2.0. Även här har Olivetti ställt en uppgradering i sikte. I så fall troligen till version 3.0. Den senare har ett mer intelligent sätt att hantera olika processers slagsmål om befintligt primärminne.

Augias torde kosta ca 3 - 400.000 kronor som den står och går i dagsläget.

Augias (imaginärdel)

Datorns namn är även det en gåva från Lars Philipson till oss. Eftersom namnet manar till en hel del eftertanke om man vet historien bakom, så kan det vara på sin plats att kortfattat berätta om vem Augias var.

Redan de gamla grekerna hade gudasagor. En av de mer kända sagorna i den grekiska mytologin handlar om Herakles och hans bragder. Denne Herakles var son till självaste Zeus. Hans mor däremot var en jordisk kvinna vid namn Alkmene. Herakles växte upp till en yngling av oerhörd styrka och fysiskt mod. Genom illistigt spel bakom kulisserna av en annan gudom - Hera, kom Herakles att utföra ett hemskt illdåd - han mördade sin hustru och sina barn.

Av oraklet i Delpi fick han veta att han för att sona sin gärning skulle tjäna konung Eurystheus lojalt i 12 hela år. Först därefter skulle han få förlåtelse och hans själ få frid. Det var på så sätt det kom sig att Herakles utförde 12 stordåd, det ena större än det andra.

Det femte av dessa stordåd gjorde att han kom i kontakt med konung Augias. Denne konung Augias hade ett stall som inte hade rengjorts på många år. Pest hade spritt sig ifrån stallet över landet Elis, där Augias var härskare, men han var för lat för att göra någonting åt saken. När Augias fick höra att Eurystheus befallit Herakles att rengöra stallet bara skrattade han, då han fick höra att det skulle vara gjort på en enda dag.

-"Det kommer att behövas en timme för varje skottkärra. Det kommer att behövas hundra - ja, kanske tusen - skottkärror, och så vitt jag vet finns det ingen dag som omfattar tusen timmar!" sade han. Han var så säker på att det var omöjligt att han lovade Herakles en del av stallets boskap som betalning om det blev gjort på en enda dag.

Herakles log litet. Han hade sett att floden Alfeios flöt förbi i närheten och hade redan tänkt ut en plan. Han byggde en damm över floden, som fick dess långsamt framflytande vatten att ta en ny riktning, så att det leddes in på stallområdet och spolade med sig allt avfall, tills inte minsta smuts fanns kvar.

-"Använd dina tusen skottkärror till att köra bort jorden från dammen, så floden återtar sitt gamla lopp", sade Herakles till Augias med en glimt i ögat.

Läsaren får själv gissa vilken symbolik man kan lägga i denna historia. Men eftersom vi inte känner oss alldeles smickrade av denna historia så ansträngde vi oss lite extra när vi forskade i den gamla grekiska mytologin. Historien var nämligen inte riktigt slut.

Augias var inte alls glad över Herakles fiffighet och vägrade betala med de kreatur han utlovat. Detta glömde inte Herakles och när han väl gjort sina 12 underverk och fått sina synders förlåtelse så återvände han med en liten arme för att hämnas på Augias.

Det bar sig inte bättre än att Augias blev dräpt och Herakles och hans här fick med sig ett rikt krigsbyte tillbaka. På den tiden var det vanligt att man offrade en del av krigsbytet till ett gott ändamål för att hålla sig väl med gudarna i framtiden. I det här fallet avsattes en del av krigsbytet till en fond vars avkastning kom att bekosta ett periodiskt återkommande evenemang - de Olympiska Spelen.

Även om vi på intet sätt vill se vår kära dator gå en för tidig död till mötes, får vi erkänna att vi skulle bli mycket stolta om den gav upphov till något som ens lokalt i E-huset blev lika känt som OS.

The History of Unix

Som titeln antyder är den här artikeln skriven på engelska. Men den är rolig, intressant och framför allt kort - så läs den ändå. Att den är skriven på engelska beror på att vi hämtat den ur tidningen BYTE (augusti 1983).

In the late 1960s, a project was underway at the Massachusetts Institute of Technology (MIT) to improve the state-of-the-art in timesharing software. Along with MIT, Bell Laboratories and General Electric (GE was once a mainframe computer manufacturer) were collaborators in the venture. But Multics, as the system was christened, was too big and slow - an overdesigned behemoth of the software world. So Bell Labs pulled their people out of the project, which left MIT and GE to develop the system further on their own. (They did, and Honywell, who later bought GE's computer operation, still sells Multics.) Unfortunately, that left Ken Thompson, a computer scientist at Bell Labs, without any hardware to run his video game.

Thompson had written a simulation of the solar system, called Space Travel, which ran on the Multics system on a timesharing terminal. The loss of Multics was the impetus he needed to find hardware he could use exclusively. He gained access to a Digital Equipment Corporation (DEC) PDP-7, complete with a video display that would enhance Space Travel tremendously. While Thompson was rewriting Space Travel for the PDP-7, he began experimenting with some ideas he had for a new type of file system. Working in PDP-7 assembly language, he soon had his file system running with some utility programs and a central core (or kernel) that together made a rudimentary operating system. Here was a system designed by one man for the sole purpose of making his own software-development work easier. Unix was thought to be a good name for it - the Uni (one) was a word play on the Multi (many) of Multics.

Unix came to the attention of others at Bell Labs, including Dennis Ritchie, another systems software designer. Together, Ritchie and Thompson enhanced Unix, adding some word-processing facilities in response to hints that another department needed a word-processor. This earned the designers enough funding for a PDP-11 minicomputer, a more modern and reliable machine than the PDP-7. Eventually, other departments bought PDP-11s and chose to use Unix for the software base rather than DEC's own operating systems.

But Thompson, dissatisfied as he was with other operating systems, also felt that programming languages could be improved. FORTRAN was tried and discarded. He then worked for a while on BCPL (Basic Combined Programming Language), which was a simplification of CPL, itself a simplification of the Algol 60 language (today we would call Algol 60 a Pascal like language). Thompson condensed BCPL down to its most basic features. The interpreted language that resulted he named simply B. Ritchie then took the best parts of B, reworked them until he had a language that was simple and elegant, added data structures, and called it C. Ritchie and Thompson both felt this was a language suitable for systems programming - one that allowed a programmer to express concepts clearly without being tied to one machine's architecture, and yet was efficient enough so that assembly language would not be needed for speed.

Getting a Handle on Portability

Unix was rewritten in C in 1973, whereupon Ritchie and Thompson realized that because C was a relatively high-level language, compilers could be written on other computers to give them C capability too. And because Unix was written in C, theoretically Unix could then be moved to these other machines. The experiment was tried in 1977 with an Interdata 8/32, a 32-bit minicomputer that was as

unlike the PDP-11 as possible. All code specific to the PDP-11 was taken out of the kernel and rewritten to make it easy to transport Unix. After the Interdata test, they moved Unix to an IBM/370 mainframe. With each trial they learned more about C, Unix, and portability in general

Until Unix, operating systems were written exclusively in assembly language. This long, error-prone process seemed the only appropriate to an industry that considered machine efficiency to be more essential than human efficiency because computers were more expensive in dollars and cents than human labor. Compared to other languages, assembly language allows the fastest execution of instructions and takes up the least memory space; therefore, programs as important as operating systems could only be written in assembly language. Who cared if a programmer or two went crazy trying to understand it? What was the difference if it took a long time to write and three times as long to debug?

Ritchie and Thompson saw that a software designer's environment was more important, in the long run, than that of the computer; computer hardware tends to get cheaper and faster, while the cost of labor in both economical and emotional terms tends to go up. This last is especially true when the software tools at hand are not appropriate for the job. Unix forever broke the notion that a system had to be written in assembly language and therefore tied to a specific computer design, word size, or architecture. For the first time, an entire programming environment, including file system, kernel, applications packages, utility programs and user interface could be moved to an entirely different type of machine.

Think about that for a moment. Look at the CP/M 2.2 operating system. CP/M has gained immense popularity; it runs on computers made by literally hundreds of different manufacturers and supports many different languages and applications packages. Why is it so popular with computer makers? CP/M is portable to many different hardware configurations. The catch is that the systems must use a microprocessor that runs 8080 assembly code.

In comparison you can now run Unix or Unix-compatible systems on computers based on: 8080, Z80, 8086, 8088, 80286, 68000, 32000, LSI-11, VAX, IBM/370 and a lot of other processors. Typical hardware configurations range from $2000 to considerably more. A program correctly written in C for any of these machines will run on any other one, needing only to be physically moved and recompiled. No doubt you can see why so many software houses have discovered Unix. By using C and Unix, they can expand their potential customer base tremendously with little trouble - one user manual, one customer support group, one version of source code. The net result can benefit everyone with better, more widely used software at lower prices.

Kort introduktion till UNIX

för dig som kan VMS

av Lars Svensson

Denna artikel är avsedd som en introduktion för den som inte tidigare kommit i kontakt med operativsystemet UNIX.

Inledning

UNIX och VMS liknar varandra ganska mycket. Om du har erfarenhet av exempelvis UNIVACs OS 1100 eller NORDs SINTRAN III kommer du att märka, att likheterna mellan UNIX och VMS är betydligt fler än skillnaderna.

(En liten brasklapp för framtiden: Alla exempel som ges, hänför sig till VMS version 3 eller senare och AT&T UNIX System V.)

Några vanliga stötestenar

Redan innan jag börjar skall jag komma ihåg att nämna ett par saker, som kan orsaka många svordomar för den ovetande.

1. UNIX skiljer på stora och små bokstäver. Detta gäller i stort sett i alla sammanhang Exempelvis är "`foo`" och "`Foo`" namn på två olika filer.

2. Om man i VMS modifierar en fil på något sätt, har man kvar den gamla filen, men med etlägre versionsnummer. Den räddningsplankan finns *inte* i UNIX. En editor modifierar alltså verkligen filen, inte en kopia av den. Detta gör som du förstår också kommandot **`PURGE`** liksom själva versionsnumret onödigt.

3. UNIX kan vara väldigt kortfattat av sig. Om du i VMS befinner dig i ett tomt directory och ger kommandot **`DIR`**, kommer du att få svaret "`No files found.`". I motsvarande situation i UNIX fås inget meddelande alls.

Kommandotolken

I VMS finns en kommandotolk som heter **DCL**. Det är det program som tolkar vad du skriver på en kommandorad och anropar rätt programsnutt för att få gjort vad du bad om. I UNIX finns det flera olika kommandotolkar (så kallade shells) tillgängliga. De flesta böcker om UNIX behandlar bara en av dessa, som tur är just den som finns på Augias, nämligen **`sh`** (Bourne shell, Bourne efter killen som skrev den). Denna shell (liksom andra UNIX-shells) tillåter inte att du förkortar kommandonamn. Detta har en orsak, som är värd några rader.

När du skall köra ett program i VMS skriver du

```
$ run fido
```


eller, om det är en kommandofil du skrivit,

```
$ @fido
```

eller, om det är ett av de program som följer med maskinen, till exempel programmet som kopierar en fil till en annan

```
$ copy fil1 fil2
```

UNIX skiljer inte på dessa tre fall. I samtliga fall skriver man programnamnet rätt och slätt, följt av eventuella parametrar. Operativsystemet håller reda på om det är en binärfil (exekverbar fil) eller en kommandofil och handlar därefter.

Om man skriver en bokstavskombination till **`sh`** gör den följande:

1. Kollar om man har nån fil med det namnet i det directory där man befinner sig, och kör i så fall denna.

2. I annat fall kollas om man har nån fil med det namnet i directoriet ovanför det där man befinner sig, och i så fall körs denna.

3. I annat fall kollas om det finns nåt systemprogram lokalt installerat på den här maskinen (till exempel nåt utskriftsprogram. Såna är ju ofta installationsberoende) med rätt namn, och i så fall körs detta. Detta innebär faktiskt att **`sh`** tittar i ett speciellt directory om där finns nån fil med rätt namn.

4. I annat fall kollas om det finns nåt "riktigt" systemprogram med det namnet på den här maskinen, och i så fall körs detta. Vid det här laget har du säkert räknat ut, att detta innebär, att **`sh`** letar i ytterligare ett antal directories efter en fil med rätt namn.

Som du märker upprepar **`sh`** ett visst handlingsmönster, men för olika directories. Användaren kan i en så kallad path specificera vilka directories som **`sh`** skall gå igenom på jakt efter rätt program. Den ordning jag skissat ovan är väl minimum.

Med den här mekanismen blir det svårt att få kommandoförkortning att fungera. Ponera att du har ett program som heter **`foobar`** i det directory där du befinner dig, och ett program vid namn **`foo`** i directoriet ovanför. Sedan skriver du **`foo`** till **`sh`**. Vad skall den välja?

Slutsatsen är: Ingen kommandoförkortning. Nästa slutsats: kommandonamn bör hållas korta.

En annan sak: Optioner i VMS anges med ett snedstreck, som i

```
DELETE/CONFIRM  ...
```

Det vanliga i UNIX är, att optioner till kommandon anges med ett streck, som i motsvarigheten till VMS-kommandot ovan

```
rm -i ...
```

Och så en liten sak som jag egentligen inte vet var jag skall stoppa in: Om ett VMS-kommando kan ta flera filnamn, skall dessa separeras med komma, ",". Detta vill UNIX inte vara med om. Filnamnen separeras med blanktecken.

In- och utmatning med mera

In- och utmatning är i UNIX hårt standardiserat. Den bärande iden är, att utmatningen från ett program i princip skall gå till på samma sätt oavsett om den går till en terminal, en fil eller direkt till ett annat program. (Samma sak gäller förstås inmatningen.) Standardiseringen gäller också programmen. Varje program har en standardiserad "in-kanal" vid namn **stdin**, en standardiserad "ut-kanal" vid namn **stdout** och en extra "ut-kanal", avsedd för felutskrifter och motsvarande, vid namn **stderr**. "Kanalerna" är väsentligen filer på samma sätt som input och output i ett pascalprogram är filer.

In- och utkanalerna hos ett program kan man "dirigera om" genom att formulera sig på lämpliga sätt redan på kommandoraden. Detta saknar direkt motsvarighet i VMS och kan därför vara värt några ord.

De flesta program skriver direkt på **stdout**. Om du inte anger något annat kommer all utskrift på stdout att hamna på skärmen. På samma sätt läser många program **stdin**. Om du inte anger något annat kommer programmet att läsa från tangentbordet.

Säg att du har ett program under VMS som läser från terminalen, och du vill ha det att i stället läsa från en fil **fil.dat**. Detta gör du då genom att, innan du kör programmet, skriva

```
$ assign/user fil.dat sys$input
```

eller liknande. I UNIX sker detta istället genom så kallad "redirection" av **stdin**. Om programmet heter **foo** och filen heter **bar** skriver du

```
foo < bar
```

Vill du att programmet skall skriva på en fil istället för på terminalen skriver du istället

```
foo > bar
```

varigenom du alltså gör en "redirection" av **stdout**. Dessa båda metoder går självklart att kombinera, som i

```
foo < bar1 > bar2
```

Här kommer alltså bar1 att "anslutas" till **stdin** hos **foo**, och **stdout** kommer att anslutas till filen **bar2**. Detta innebär att utskriften från programmet hamnar på **bar2**.

Även det tredje fallet, där man vill låta ett program läsa som input vad ett annat program har skrivit, kan formuleras på en kommandorad. Härvid används pipe-operatorn, "|" ("ö" i en svensk teckenuppsättning). Skriver man

```
foo | bar
```

kommer utskriften från programmet **foo** att användas som indata till programmet **bar**; man har alltså "anslutit" **stdout** hos **foo** till **stdin** hos **bar**. (Detta görs i VMS med en temporär fil, som **foo** får skriva och bar sedan läsa.)

Förgrund respektive bakgrund

I VMS finns det tre alternativa "moder" i vilka ditt program kan köras, nämligen interaktivt jobb, via kommandot **SPAWN** eller som batchjobb via kommandot **SUBMIT**. I UNIX skiljer man på förgrund och bakgrund. Förgrunden motsvarar den interaktiva moden i VMS. Du ger kommandot, maskinen tolkar ditt kommando, maskinen kör det program som utför ditt kommando, och först därefter får du chansen att ge ett nytt kommando. Ett kommando kört i bakgrunden motsvarar ungefär en **SPAWN/NOWAIT** i VMS; du behöver alltså inte vänta på att kommandot skall exekveras färdigt innan du ger nästa kommando. För att **sh** skall köra ditt kommando i bakgrunden räcker det att du sist på kommandoraden skriver ett "**&**". Programmet kommer att snurra vidare tills det blir färdigt eller du loggar ut. Vill du att det ska fortsätta även om du loggar ut skriver du istället:

```
nohup kommando &
```

Eventuella utdata från en sådan körning hittar du sedan i filen nohup.out.

Det finns möjligheter att stoppa ett program som man börjat köra i förgrunden. Det går bra att ha flera program igång samtidigt; max ett kan ligga i förgrunden, men flera kan vara stoppade och flera körande i bakgrunden samtidigt. Kommandona för styrningen av detta beskrivs i manualen för **sh**. Ett av dem:

```
^Z  (ctrl-Z)
```

stoppar det program som körs i förgrunden.

Filen i UNIX

I UNIX finns det i motsats till i VMS bara en filtyp. Alla filer är organiserade som *file of byte*. Detta gäller i princip även directories och sk periferifiler. En periferifil är till exempel en terminal eller en skrivare, så som ett program ser den. Jämför stycket om in- och utmatning ovan!

En vanlig typ av fil är textfilen. Som alla filer i UNIX är den alltså organiserad som "file of byte". Av historiska skäl markeras slutet på en rad i en textfil med ett linefeed (alltså inte med ett carriage return, vilket kanske skulle kännas mer naturligt.) Detta innbär inga ändringar för dig, när du knackar in text; du kan använda **<cr>**-tangenten precis som vanligt. Det enda som påverkas är den interna representationen i filen.

Filsystemet

Filsystemet i UNIX är mycket likt det i VMS. På samma sätt som i VMS kan du skapa dina egna underdirectories i så många nivåer du har lust till. Liksom i VMS antas en fil ligga i ett "default directory", om man inte specificerar något annat.

Adresseringen av en fil är lite annorlunda. Vill du specificera en fil **foo** i ditt default-directory, skriver du

```
foo
```

Vill du specificera en fil **bar** i directoryt **foo** som i sin tur ligger i ditt defaultdirectory, skriver du

```
foo/bar
```

Precis som i VMS är alltså filsystemet uppbyggt som ett träd. Ett antal konventioner underlättar för användaren att hitta i trädet.

Trädets rot är filen **/**. I detta directory ligger ett antal olika filer, bland dem **/bin, /dev, /lib, /tmp /usr** och **/usr2**. **/bin** innehåller systemprogram i exekverbar (binär) form. **/dev** innehåller periferifilerna till alla systemets periferienheter (devices). **/lib** innehåller bibliotek (libraries) till kompilatorer med mera dylikt. I **/tmp** kan platskrävande program som kompilatorer lägga upp temporärfiler. Under **/usr** och **/usr2** slutligen finns de olika användarnas privata filer. På Augias är **/usr** reserverad för systemets olika användaridentiteter medan vanliga användare huserar på **/usr2**.

Det är brukligt att varje användares eget filträd i viss mån avspeglar detta mönster. Det är mycket lättare att hitta bland sin arbetskamrats program om man vet att han har dem i sitt bin-directory.

Lägg märke till, att begreppet filtyp inte finns i UNIX. En punkt, ".", kan förekomma var som helst i ett filnamn. Det är ganska vanligt, att man låter filer som innehåller c-program sluta med "**.c**", pascalfiler med ".p" och så vidare, men detta är egentligen bara konventioner. Somliga program kan dock vara petiga med suffix. C-kompilatorn **cc** är ett exempel.

En annan sak med "**.**" i filnamn: Om du som första tecken i ett filnamn har en punkt, kommer filen inte att synas vid vanliga directorylistningar. Detta är avsett för filer som man inte ändrar i så ofta. Så heter till exempel motsvarigheten till VMS-filen "**LOGIN.COM**" "**.profile**" i **sh**.

Wildcards

Vill du ta bort alla filer i ditt defaultdirectory, skriver du i VMS

```
DEL  *.*;*
```

I UNIX har man ju ingen explicit filtyp och inget versionsnummer på filerna. De tre delfälten i filnamnet, villka alla måste matchas med var sin "*****", smälter samman till ett fält, och man får

```
rm *
```

En "*****" expanderas av **sh** till en lista, bestående av alla filnamn i det aktuella directoriet. Liksom i VMS kan man skriva "***.n**" för att matcha bara de filer som har ett namn som slutar med "**.n**". En "*****" matchar noll eller flera tecken. Ett frågetecken "**?**" matchar ett tecken. Uttrycket "**[abc]**" matchar endera av tecknen "**a**", "**b**" och "**c**". Uttrycket "**[a-z]**" matchar ett styck liten bokstav.

Ett exempel på en dialog: (**ls** motsvarar **DIR**, **rm** motsvarar **DEL**. "**$**" är **sh**-s prompt och motsvarar alltså "**$**" i VMS. Man kan för övrigt välja sin prompt själv.)

```
$ ls

#intro.n
carne.n
carne.pr
sfs.n
taxes.n
intro.n
intro.n.out
```



```
uppsk.t
uppsk.n
uppsk.s

$ rm #*

$ ls

carne.n
carne.pr
sfs.n
taxes.n
intro.n
intro.n.out
uppsk.t
uppsk.n
uppsk.s

$ rm carne.*
$ ls

sfs.n
taxes.n
intro.n
intro.n.out
uppsk.t
uppsk.n
uppsk.s

$ ls u*.?

uppsk.t
uppsk.n
uppsk.s

$ rm uppsk.[nt]
$ ls

sfs.n
taxes.n
intro.n
intro.n.out
uppsk.s

$ rm *
$ ls
$
```

Gå gärna igenom exemplet ovan och kolla att du har förstått det!

Kommandonamn

Kommandon i UNIX heter förstås inte detsamma som i VMS. Eftersom man enligt ovan strävat efter att hålla dem korta, kan de dessutom verka ganska kryptiska. En liten lista över några av de vanligaste kommandona följer här.

cat (concatenate and print) Hakar på filer efter varandra i en lång rad. Detta sköts i VMS av **COPY**-programmet. Utskriften hamnar på **stdout**, vilket blir skärmen om man inte gör redirect. Detta gör att **cat** även är en motsvarighet till **TYPE**-kommandot. Se dock **more** nedan.

more är ett program, som man använder i stort sett vid varje tillfälle, när man i VMS skulle ha använt **TYPE**. För att man ska slippa sitta och passa med "no scroll"-tangenten för att inte missa något, fylls bara en sida av skärmen. Härefter stoppas utskriften, tills man trycker på space, varvid en ny sida fås upp. Om ett program , exempelvis med namnet **foo**, kan förväntas producera mycket utskrift, kan det vara lönt att först som sist pipe-a utskriften genom **more**, alltså

```
foo | more
```

Den vanliga användningen är

```
more file1 file2 ...
```

less Kan lite mer än **more**, men finns inte på alla maskiner. På Augias är dock läget i skrivande stund att endast **less** finns - **more** saknas således.

rm (remove) är som du kanske redan försått motsvarigheten till **DELETE**.

cp (copy) Kopierar en fil till en annan. Motsvarar förstås **COPY** i VMS.

mv (move) En motsvarighet till **RENAME** i VMS. Byter alltså namn på filen. Om man som sista argument anger ett existerande directory, flyttas de tidigare angivna filerna till detta directory.

mkdir (make directory) Skapar ett nytt directory.

rmdir (remove directory) Raderar ett directory.

cd (change directory) motsvarar **SET DEFAULT** i VMS, men klagar om man specificerar ett icke-existerande directory.

pwd (print working directory) motsvarar **SHOW DEFAULT** i VMS.

ls (list) motsvarar **DIR** i VMS. Olika optioner till detta kommando finns. En av de mera användbara kombinationerna är

```
ls -ls
```

som förutom en namnen på filerna ger storlek i diskblock och i byte, filskydd, ägare av filen samt filens ålder. Detta motsvarar ungefär VMS-kommandot **DIR/SIZ/DATE/PROT**. För att se de i normala fall osynliga filer som börjar med "." använder man optionen **-a**.

grep (get reg . expression) i a kommando motsvarar ungefär kommandot **SEARCH** i VMS, men vill givetvis ha sina kommandon i motsatt ordning. (Murphy sover aldrig.) **grep** söker efter en sträng i ett antal filer och skriver ut var den hittar strängen. (Ett osedvanligt nyttigt kommando, för övrigt!) Syntaxen är

```
grep string files
```

Det finns flera kommandon som är släkt med grep, tex **fgrep**, **egrep**, **bgrep**...

man (manual) motsvarar tillsammans med

apropos ungefärligen VMS-kommandot **HELP**. Om du vill veta vad det finns för kompilatorer på din maskin, kan du skriva

```
apropos compiler
```

varvid hela manualbiblioteket gås igenom, och alla manualsidor som har ordet "compiler" i sin rubrik pekas ut. (Delar av ord går också bra.)

```
man cat
```

kommer att ge dig hela manualen för kommandot **cat**. Manualerna är ofta långa, och i så fall körs de automatiskt genom **more**. Här måste du emellertid ge exakt rätt namn på kommandot. Om du inte hittar kommandot du är ute efter, gör **apropos** för att få veta vilket nyckelord du skall söka på.

Lägg märke till, att den trädstruktur som finns i VMS-hjälpsystemet inte finns i UNIX' manualsystem. Som orutinerad användare känner man sig för det mesta tryggare i ett VMS-system.

OBS! På Augias finns i skrivande stund varken **man** eller **apropos**. Men vi jobbar på det! Istället finns ett kommando som både heter och fungerar som **HELP** i VMS. Denna **help** är dock begränsad till ett relativt fåtal kommandon.

mail är en mailer som är ganska lik VMS-mailern, men förstås betydligt mera tystlåten. "**Type ^C to stop, ^Z when ready**" ser man inte skymten av, utan man gör bäst i att själv veta när man ska börja knappa.

who (who is on) och

w (who is on and what they are doing) är ett par olika kommandon som talar om vem som är inloggad, och som alltså grovt motsvarar VMS-kommandot **SHOW USER**. Av historiska skäl finns det flera olika kommandon, som gör i stort sett samma sak. Prova alla, använd det du trivs med. På Augias visar **w** inte vad användarna gör, utan istället lite utförligare info om vad de heter.

Editorer

I VMS heter den vanliga editorn **EDT** eller **TPU** (från och med version 4.x). Dessa editorer är mycket beroende av en bra terminal, till exempel en vt100, för att kunna ge sitt bästa. I UNIX heter standardeditorn **vi** (visual). Detta är en mycket kraftfull editor, men lite knölig om man är van vid till exempel **EDT**. Dess fördel är att den kan användas med (nästan) vilken terminal som helst.

Editorn vi är egentligen byggd "ovanpå" en radeditor, som heter **ex**, som i sin tur är en utveckling av en tidigare radeditor vid namn **ed**. Liksom **EDT** har en kommandomod (nås genom **GOLD-7** eller **^Z**) har vi en kommandomod. Denna mod är identisk med radeditorn **ex**.

På många UNIX-system finns det fler editorer att välja på. En vanlig "supereditor" är **emacs**, som egentligen är flera olika editorer med likartat uppförande och samma namn. Deras största fördel är kanske deras utbyggbarhet, som ger förmåga att imitera andra editorer. Härigenom behöver man inte lära om alldeles när man flyttar till ett UNIX-system.

Avrundning

Ett par rader om UNIX från en lite annan synvinkel får avsluta denna artikel.

UNIX kan vara lite stökigt för en nybörjare. Tystlåtenheten, de kryptiska kommandonamnen, det lite ovanliga systemet med **stdin** och **stdout**, den genomgående metoden att skriva massor av kommandon på samma kommandorad (med "|" emellan), svårtydbara felutskrifter med mera kan skrämma upp den som jobbat med ett lite "mjukare" system som VMS. Efter en tids användning kommer man vanligen över den första motviljan.

Man kan säga, att UNIX är ett operativsystem huvudsakligen avsett för programmerare. Programutveckling underlättas oerhört av den tidigare nämnda standardiserade in- och utmatningen. Ett otal mycket praktiska program, som utför ofta återkommande uppgifter som till exempel utbyte av stora bokstäver mot små, sökning i filer efter en viss sträng, hantering av databaser med mera följer med systemet vid köp. Vidare finns det gott om kompilatorer för olika språk; en kompilator för språket C följer med operativsystemet, liksom en lisp-interpretator och ett kompilatorutvecklingssystem etc.

En lösryckt replik om UNIX: "You may dislike it at first, but after using it for a year or so, you'll be the local wizard."

Litteraturtips

"The UNIX System" S R Bourne, Addison-Wesley 1982. En bra bok om UNIX, som inte kräver annat av läsaren än en viss aning om datorer i allmänhet.

"The UNIX Programming Environment" Kernighan & Pike, Prentice-Hall 1984. En förträfflig bok, som "leder" läsaren från klarhet till klarhet. Kan icke nog rekommenderas! En och en halv nivå över den föregående boken.

"The C Programming Language" Kernighan & Richie, Prentice-Hall 1978. Behandlar språket C, vilket man inte kan undvika att komma i kontakt med om man använder UNIX.

Liten Lathund

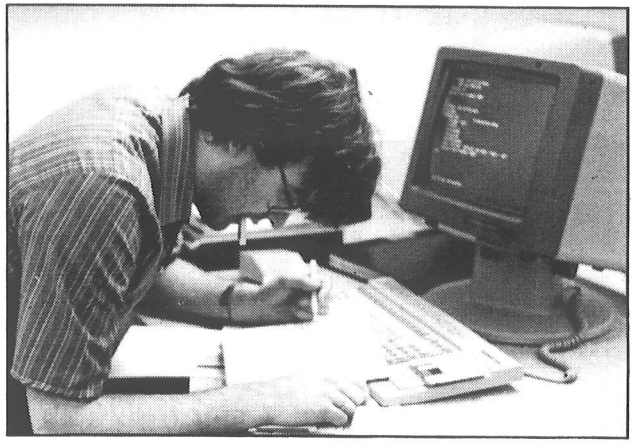
VMS-kommandot:	motsvaras av UNIX kommandot:
COPY SYS$INPUT nisse.txt	cat > nisse.txt
COPY nisse.txt sture.dat	cat nisse.txt > sture.dat
COPY nisse.txt sture.dat	cp nisse.txt sture.dat
CREATE nisse.txt	cat > nisse.txt
CREATE/DIRECTORY [.pascal]	mkdir pascal
DEBUG	sdb
DELETE	rm
DELETE pascal.dir	rmdir pascal
DELETE/CONFIRM	rm -i
DIFFERENCES	diff
DIR/SIZE/DATE/PROT	ls -ls
DIRECTORY	ls -a
DUMP	od -x
EDIT nisse.pas	vi nisse.p
HELP	man man
HELP COBOL	apropos cobol
HELP COBOL	man cobol
LOGOUT	exit
MACRO	as
MAIL	mail
PHONE sture	write sture
PRINT fil	lpr fil
RENAME gammal ny	mv gammal ny
RUN upg3	upg3
RUNOFF	nroff
SEARCH *.pas "AddMatrix"	grep AddMatrix *.p
SET DEF SYS$SYSROOT	cd /
SET DEF [-.pascal.upg3]	cd ../pascal/upg3
SET DEF [.pascal.upg3]	cd pascal/upg3
SET DEFAULT SYS$LOGIN	cd
SET DEFAULT [-]	cd ..
SET DEFAULT [.pascal]	cd pascal
SET HOST	login
SET HOST mojtnamn	rlogin mojtnamn
SET PASSWORD	passwd
SET PROTECTION	chmod
SHOW DEFAULT	pwd
SHOW PROCESS	ps
SHOW SYSTEM	ps -f
SHOW USERS	who
SORT infil utfil	sort infil > utfil
SPAWN/NOWAIT kommando	kommando &
STOP	kill -9
SUBMIT filnamn	filnamn
TYPE nisse.txt	cat nisse.txt
TYPE nisse.txt	more nisse.txt
^Y	^C
^Z	^D

Fotokatalog

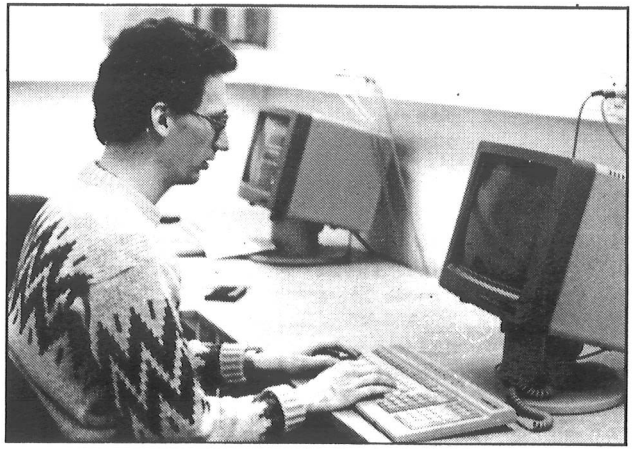
LTH:s Datorförening bildades våren 1985. Syftet var att verka för att vi skulle få ett stort datorsystem till Lund som skulle drivas helt av de studerande själva. Vi fick också några stora, begagnade datorer, men skolan visade sig svårflörtade när det gällde lokaler mm. Vårt mål är fortfarande att driva egna datorer i egna lokaler. Däremot har vi tänkt om lite när det gäller deras fysiska storlek.

Augias är inte helt och hållet vår egen dator, och vi har fortfarande inga lokaler. MEN med Augias har vi fått chansen att visa att det finns elever som är intresserade av att köra på en sådan dator.

Därför har vi som sitter i styrelsen lagt ner en hel del tid de senaste månaderna på att lära oss Unix och Augias. Vi hälsar dig välkommen dels till våra kurser, dels att komma med frågor. En av styrelsemedlemmarna saknas på bilderna. Det är Anders Landin, D-84. Han kan nås via mail från valfri VAX på skolan om ni anger adressen så här: `perdix"augias!anders"` Så här ser vi andra ut:



Ulf Andersson, D-85
`perdix"augias!ulf"`



Jörgen Hägg, E-83
`perdix"augias!jh"`