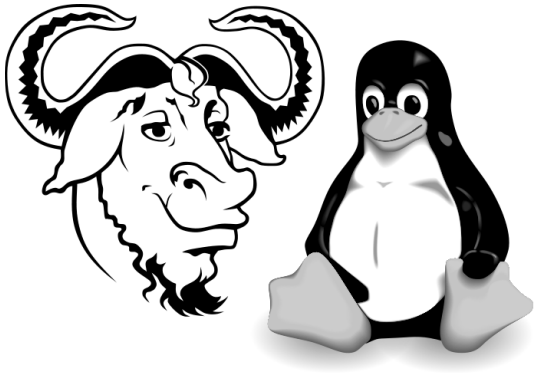
Kapitel 2

UNIX och GNU/Linux



2.1 Historik

Linux är inte så nytt. Det bygger vidare på traditionen från UNIX, men inte på dess källkod. Första versionen av UNIX kom 1969. Det utvecklades av Ken Thompson och Dennis Ritchie på Bell Labs. 1972 skrevs UNIX om i det av Ritchie skapade språket C. På 70-talet och början av 80-talet fick universitet använda UNIX gratis, inklusive källkod. Men sen försökte AT&T, som äger Bell Labs, att ta betalt för UNIX.

På universitetet i Berkeley utvecklades, som motreaktion mot kommersialiseringen, en friare version kallad BSD, Berkeley Software Distribution. På MIT jobbade då Richard Stallman, han tyckte att inte heller BSD var tillräckligt fritt.

BSD-licensen säger att källkoden även får användas kommersiellt.

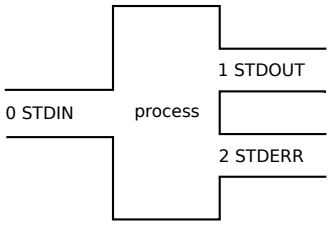
Stallman gillade inte att något fritt kunde övergå till ofritt, så 1984 startade han GNU-projektet. GNU betyder GNU's not UNIX (det är en så kallad rekursiv förkortning, som hänvisar till sig själv) och dess licens heter GPL (The GNU General Public License). GPL ger användarna frihet att använda, studera, distribuera och förbättra koden till alla program. Däremot får man inte hindra andra att få samma friheter. Kod släppt under GPL måste fortsätta vara GPL.

GNU-projektet gjorde egna versioner av de flesta av tillbehören i UNIX, men de lyckades inte skriva en bra kärna. 1991 skrev Linus Torvalds första versionen av Linuxkärnan, som passade bra att använda i GNU-systemet. Ska man vara petig bör man därför kalla systemet för GNU/Linux (som i denna boks titel), för med Linux menar man bara kärnan som ligger närmast hårdvaran. Men även jag slarvar ofta med det, och mycket i moderna distributioner kommer inte från GNU, utan är släppt under andra mer eller mindra fria licenser. Och jag kan inte räkna upp alla.

UNIX var redan från början ett fleranvändarsystem. Många kommandon som finns i nutida GNU/Linux fanns med redan på 70-talet. Filosofin bakom UNIX är bland annat att varje program ska göra en sak, men det ska göra det bra. Man kan koppla ihop program så att utdata från ett blir indata till ett annat.

2.2 Skal

Vad är ett skal? Det heter ”shell” på engelska och är den process som tar emot dina kommandon. Exempel är Bash, Z Shell (zsh) och tcsh. Det är skalet som du använder mest, både i terminalfönster och vid SSH-inloggning. Det har ett mindre antal inbyggda kommandon, och kan starta alla andra program. Man kan även skriva program som exekveras av skalet, mer om det i kommande kapitel. De olika skalen har lite olika syntax, jag kommer mest beskriva `/bin/bash` i den här boken.



Figur 2.1: *STDIN och STDOUT*

2.3 Omdirigering och rör

Varje process har alltid (minst) tre ”filer” öppna. Se figur 2.1. Processen läser från STDIN (standard in), som kan vara ett tangentbord, en fil eller ett annat program. Både STDOUT och STDERR är för utdata, men med skillnaden att standard out är för normal utdata och standard error är för felmeddelanden. Siffrorna i bilden ovan är fildeskriptorer och är alltid desamma. För att dirigera om STDOUT från terminal till en fil använder man tecknet > (det går även med 1>). Vill man däremot endast ha felmeddelanden (STDERR) i filen använder man 2>. Vill man bara lägga till i slutet av filen och inte skriva över hela använder man >> och 2 >>. Till exempel:

```
cd /etc; grep kalle * > /tmp/kalle.txt 2> /tmp/fel.
```

Det kommandot kommer att spara alla rader som innehåller ordet kalle i filen `kalle.txt`, och i filen `fel` hamnar en lista med alla filer som en vanlig användare inte får läsa (t.ex. `/etc/shadow`).

Med en pipeline (tecknet |, rörledning på svenska, eller kort och gott pipe) kopplar man ihop STDOUT från en process med STDIN på en annan. T.ex. `ps aux|grep httpd` för att visa alla webbserverprocesser. (Jag bygger vidare på det exemplet i 12.8). Man kan ha flera pipes efter varandra, mer om det kommer i kapitel 11 om skript. I UNIX (och Linux) kör processerna på varje sida om | helt parallellt, därför fungerar saker som:

```
tail -f /var/log/httpd/access_log|grep bild.jpg
```

2.4 Alternativ

Linux är inte det enda fria systemet. Vill man köra BSD (som jag nämnde i början av detta kapitel) finns det tre stora att välja på: FreeBSD, NetBSD och OpenBSD.

Lite förenklat är FreeBSD inriktat på storskalig serverdrift, OpenBSD på säkerhet och nischen för NetBSD är att det ska gå att köra på så många olika sorters datorer som möjligt. Huruvida det är rätt att tvinga på andra frihet, som GPL, eller inte bry sig om att andra använder koden till proprietära (kommersiella) system, som BSD tillåter, är för många en viktig fråga. Men det finns bra saker med båda licenserna. Båda har fritt tillgänglig källkod, och är ofta gratis.

Vill man vara ännu mer lik klassisk UNIX kan man använda Open Solaris. Det har ett avancerat filsystem som heter ZFS. Tyvärr är inte pakethanteringssystemet lika bra i Solaris (än). Och vissa andra saker känns lite gammalt. Solaris utvecklades av Sun Microsystems, som januari 2010 köptes upp av Oracle.

Andra system som är mer lika ursprunglig UNIX, och med sluten källkod är: AIX från IBM, HP-UX från Hewlett Packard, Tru64 (ursprungligen från Digital Equipment Corporation, men de ägs nu av HP). Även Mac OS X är baserat på UNIX (via Mach, FreeBSD, NetBSD och NeXTStep).

För smarta telefoner och surfplattor används Android och MeeGo/Maemo, båda har en Linuxkärna i botten och har mestadels öppen källkod. Men dem skriver jag inte mer om här eftersom boken främst handlar om serverdrift.

2.5 Linuxdistributioner

GNU/Linux finns i många varianter, kallade distributioner. De kan delas in efter vilket paketsystem som används: deb, rpm eller annat. Stor bland de RPM-baserade är Red Hat Enterprise Linux (RHEL), som kostar pengar att köpa, men där källkoden är fri. CentOS använder Red Hats kod och byter loggan till sin egen – helt tillåtet enligt GPL. Man får därmed samma produkt som RHEL gratis via CentOS. Men CentOS har inte lika bra support som Red Hat. Fedora kan ses som Red Hats utvecklingsplattform. Många, men långtifrån alla, av utvecklarna av Fedora jobbar för Red Hat. RHEL kommer

som ny version vartannat eller var tredje år, Fedora däremot släpps två gånger per år, och med massor av uppdateringar däremellan. Även Mandriva och SUSE använder RPM.

En populär distributionen som är baserad på Debians paketsystem (där filerna slutar på .deb) är Ubuntu. Ubuntu kommer med en ny version två gånger per år (april och oktober). Debian kommer mer sällan, men de lånar kod från varandra. Mest är det Ubuntu som lånar från Debian. En annan populär Debian-variant är Linux Mint. Mint är lite mer konservativ när det gäller grafiskt gränssnitt, många tycker att Ubuntu och Fedora har ändrat för mycket med Unity och GNOME 3.

Det finns även andra system som antingen kompilerar allt från källkod (Gentoo m.fl) eller har annat paketsystem (Slackware m.fl), men de används inte lika mycket numera. Se tabell 2.1 för några exempel.

För mer info se *<http://distrowatch.com/>*.

Tabell 2.1: Linuxdistributioner

RPM	deb	annat
RHEL	Debian	Gentoo
CentOS	Ubuntu	Slackware
Fedora	Kubuntu	Arch
SUSE	Mint	rPath

Likheterna mellan distributioner är större än skillnaderna. Är de ungefär lika gamla är det ofta liknande versioner av de medföljande paketen. Dock kan det vara bra att veta hur man använder både rpm/yum och dpkg/apt-get, så jag berättar lite om dem. Lär dig gärna mer om den variant som du inte brukar använda, man vet inte när man behöver kunna rpm om man är van vid apt, eller tvärtom.

2.6 RPM och Yum

RPM betydde först Red Hat Package Manager, men numera utläses det RPM Package Manager.

Är filen man vill installera sparad i den katalog där man står skriver man:

```
rpm -ivh httpd-2.2.15-1.fc13.x86_64.rpm
```

Det går även att lägga in från en URL:

```
rpm -ivh http://ftp.df.lth.se/pub/centos/5/os/i386/
CentOS/bc-1.06-21.i386.rpm
```

(hela raden ska skrivas, den är bara radbruten här i boken). I argumenten till rpm betyder `i` install, `v` verbose och `h` hash (skriver ut # så man ser att paketen installeras).

Det händer dock att vissa paket kräver andra paket. När enbart `rpm` fanns så var det jobbigt att behöva ladda ner alla paket, och veta vilka som behövdes. Därför skapades yum, som alltså är en påbyggnad till rpm. Till yum finns en förutbestämd lista med adresser att ladda ner från, repositories eller kort och gott repos. På svenska kan det heta paketlager.

För att installera webbservern Apache skriver du `yum install httpd`. Bara så, utan att ange version eller annat i filnamnet. Man får automagiskt den senaste, och behöver man andra paket för att det man installerar ska fungera (beroenden/dependencies) så sköter yum även det.

Både yum och rpm använder samma databas, så det blir inga problem med att installera olika paket med rpm respektive yum. Trots att yum finns så finns det saker som är bättre att göra med rpm. Till exempel ger `rpm -qa|sort` en alfabetisk lista över alla paket, `rpm -qi bash` ger detaljerad info om paketet bash, `rpm -qf /etc/sudoers` berättar vilket paket den filen tillhör, `rpm -e httpd` avinstallerar httpd (Apache) och med `rpm -V bash` kan man se om någon av skalets filer har ändrats sen installationen.

Även yum kan användas till annat än att installera, med `yum update` uppdaterar man alla installerade paket om det finns nyare i den repo man använder. För att avinstallera kan man använda `yum remove httpd`. Vill man installera hela grupper av program (t.ex. GNOME) kan man först köra `yum grouplist` och därefter:

```
yum groupinstall 'Textbaserat Internet'
```

(Ja, den förstår svenska, men glöm inte citattecken om det finns mellanslag.)

2.7 dpkg och apt-get

I Debian och Ubuntu används paket med ändelsen .deb. Det äldre kommandot för att installera är dpkg och det nyare med källor på nätet heter apt-get (Advanced Packaging Tool). APT kom före Red Hats motsvarighet yum. Vill man installera Apache skriver man `apt-get install apache2` vilket gör att eventuella beroenden hämtas automatiskt. Vill man se vilka paket som är installerade kan man köra `dpkg -l`. Vill man söka efter nyckelord, till exempel alla paket som har med PHP att göra, kan man använda `apt-cache search php`. Den söker även i beskrivningarna för paketen. Vill du enbart söka i paketnamnen använder du `apt-cache -n search php`. I APT heter paketen oftare något med huvudversionsnummer, som apache2 och php5 (men inte för alla paket, och det finns några sådana även för rpm).

APT letar i repos som finns listade i `/etc/apt/sources.list`. Vill du uppgradera alla installerade paket kör du först `apt-get update`. Den uppdaterar databasen med nya versionsnummer. Därefter kör du `apt-get upgrade`. En skillnad mot RPM är att med APT kan man ofta få frågor om konfigurering vid installation och uppgradering. I RPM sparas nya filer istället med ändelserna .rpmnew och .rpmsave (om de ersätter den gamla).

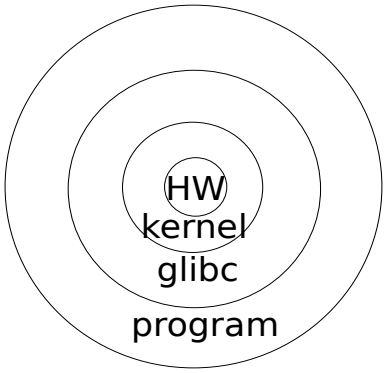
Huruvida man föredrar APT eller RPM är till stor del en vanesak, men det kan vara bra att känna till grunderna i båda.

2.8 Struktur

Efter denna utvikning om distributioner och paketformat återvänder jag till filosofin bakom fri och öppen mjukvara. Lite repetition: Linux är kärnan, den ligger närmast hårdvaran och sköter tilldelning av resurser som CPU, minne och lagring. Program kan via systemanrop be kärnan om saker som enbart den kan göra. Men det finns även mellanlager för att förenkla för de som skriver program. Ett viktigt sådant mellanlager är glibc. Även ”Gammal-UNIX” har ett libc, och g:et står förstås för GNU. C:et är samma som i programmeringsspråket C. Rutinerna går att anropa från andra språk, men av tradition har C stark koppling till

UNIX/Linux, det var för att skapa version 4 av UNIX som språket C utvecklades (1973 av Dennis Ritchie).

I GNU/Linux är hela kärnan och många systemprogram skrivna i C. Glibc och andra bibliotek laddas dynamiskt av respektive program. Systemanrop är beskrivna i sektion 2 av manualen och biblioteksanrop i sektion 3. Är man programmerare kan man till exempel se `man open` (systemanrop) och `man fopen` (glibc) för att jämföra vilken nivå de olika sakerna sker på. Många biblioteksanrop använder i sin tur en mängd systemanrop. Man brukar jämföra med lagren på en lök, se bild 2.2. Innerst ligger hårdvaran, runt den kärnan med sina systemanrop (syscalls), utanför den glibc, utanför den andra bibliotek och ytterst de olika programmen.



Figur 2.2: *Löklagermodellen*

Om man räknar in bibliotek för grafik och annat kan det bli många ytterligare lager, men denna bok är mest om serverdrift. Till skrivbordssystemen är GNOME skrivet i C och KDE i C++, (men jag vill inte rita alla tänkbara löklager för de systemen . . .)

Löklagerbilden är en bra metafor att minnas, den underlättar för att förstå vad som händer var. En process är förenklat uttryckt ett körande program, och ”user space” kallas det läge som vanliga processer körs i. Där kan, eller behöver, de inte göra allt som kärnan (kernel space) kan göra. Detta styrs av flaggor i processorn, och skyddar mot misstag. Processer i userspace har ingen egen tillgång till hårdvaran. Allt sådant måste ske via systemanrop till kärnan. Till exempel kan inte en process skriva över minne som används av en

annan process (utom i vissa fall där de delar minne för att kommunicera).

2.9 Processer, fork och exec

Vad är en process? Den frågan kan besvaras på flera sätt. Ett program som körs är ett enkelt svar, på samma vis som bakande av en tårta är en process som leder fram till ett resultat. Men ett program kan ha flera processer i gång parallellt, se t.ex. om Apache i kapitel 9 hur webbservern har en huvudprocess och flera barnprocesser. En process kan också ha flera trådar inom sig.

Som kärnan ser processer består de bland annat av: utrymme i minnet, ett antal öppna filer, en ägare, en starttid och inte minst ett ProcessID (PID). Varje PID tilldelas ofta i nummerföljd, och är alltid unika. Två olika processer kan aldrig ha samma PID samtidigt. PID=1 heter init, och är den första riktiga process som startas (Det fungerar lite annorlunda med systemd eller Upstart, jag beskriver här den mer traditionella System V init). Processen init är förälder eller farförälder till alla andra processer. Varje gång en process startar (utom init) måste en befintlig process använda systemanropet fork() för att skapa en kopia av sig själv. Denna får ett annat PID, och PPID (Parent Process ID) sätts till förälderns PID. Med kommandona `pstree` och `ps auxf` kan du se vilka som är barn till vilka processer. En fördel med fork() är att det går snabbt att skapa en process med samma uppgift som föräldern, men ofta ska barnet göra något annat. För att byta uppgift används flera olika systemanrop av typen exec(), dessa byter ut programkodutrymmet i minnet och gör andra saker så att till exempel `bash` kan få en `bc` till barn, eller att en `sshd` kan ”forka” en `bash`. Jag beskriver inte detta mer detaljerat, men att känna till processer på den här nivån underlättar felsökande och förståelse.

Kärnans tabell över processer innehåller även uppgifter om hur länge en process har använt CPU. Och vilket tillstånd processen befinner sig i just nu. Det kan man se under STAT i `ps aux` eller kolumnen S i `top`. De allra flest processer sover för det mesta, state=S. Några är R=Running. Nån kan vara D=Dead eller Z=Zombie. En Zombie är en barnprocess där föräldern har avslutats för tidigt. Zombier adopteras av init och tar inte längre resurser, men att ha sådana

tyder på problem i systemet. En av kärnans viktigaste uppgifter är att skifta rätt processer till Running. Behöver någon process vänta på indata så ska en annan få chans att använda den lediga tiden. Processer har en dynamiskt tilldelad prioritet (beroende på när de körde senast, hur mycket annat som vill köra osv). Administratören kan även ange ett Nice-värde som används som utgångspunkt för den av kärnan använda prioriteten. Nice kan variera mellan –19 och +19. Standardvärdet är 0 och positiva tal betyder snällare än negativa. Nice=10 kan vara lagom för en kompilering i bakgrunden som inte ska störa processer med 0, men ändå få mer CPU-tid än processer med Nice=19. Enbart root kan sänka ett Nice-värde, men alla användare kan höja för sina egna processer. Antingen genom att skriva `nice` framför kommandot när det startas, eller använda `renice` eller trycka **r** i `top`.

2.10 Lästips

- För UNIX och Linux historia rekommenderar jag: Salus, Peter (2008), *The Daemon, the Gnu & the Penguin*. (Den finns även på nätet).
- Även hans äldre bok är läsvärd: Salus, Peter (1994), *A Quarter Century of UNIX*. Addison-Wesley.
- En ännu äldre men fortfarande läsvärd bok är: Kernighan & Pike (1984), *The Unix Programming Environment*. Prentice Hall.
- Brian Kernighan har även skrivit en ny (2011) bok om hårdvara, mjukvara och nätverk. Den heter *D is for Digital* och har undertiteln ”What a well-informed person should know about computers and communications”. Den kanske är för lätt för dig, men mycket läsvärd som allmänbildning.