# The Linux Kernel HOWTO

Brian Ward, bri@blah.math.tu-graz.ac.at, svensk översättning: Linus Åkerlund, uxm165t@tninet.se
v0.80, 26 May 1997, svensk översättning 21 juni 1998

Detta är en detaljerad guide till kärn-konfigurering, kompilering, uppgraderingar och problem-lösning för ix86-baserade system.

## Innehåll

# 1 Inledning

Borde du läsa det här dokumentet? Tja, se efter om du har något av följande problem:

- "Arg! Det här wizzo-46.5.6-paketet säger att det behöver kärn-version 1.8.193 och jag har fortfarande version 1.0.9!"
- Det finns en enhets-drivrutin i en av de nyare kärnorna som du bara måste ha
- Du har verkligen inte en aning om hur man kompilerar en kärna
- "Är de här grejerna i READMEn *verkligen* allt?"
- Du kom, du försökte, det fungerade inte
- Du behöver något att ge till folk som insisterar på att be dig att installera deras kärnor åt dem

## 1.1 Läs det här först! (Jag menar det)

Några av exemplen i det här dokumentet förutsätter att du har GNU `tar`, `find` och `xargs`. De är ganska standard; detta ska inte skapa några problem. Det förutsätts också att du känner till ditt systems filsystems-struktur; om du inte gör det så är det viktigt att du har en avskrift av `mount`-kommandots utdata, under normala förhållanden (eller en listning av `/etc/fstab`, om du förstår den). Denna information är viktig och ändras inte, förutsatt att du inte partitionerar om din hårddisk, lägger till en ny, installerar om ditt system eller något liknande.

Den senaste ”production”-versionen av kärnan, när detta skrivs, var 2.0.30, vilket innebär att referenserna och exemplen korresponderar till den utgåvan. Även om jag försöker göra det här dokumentet så versionsoberoende som möjligt, så är kärnan ständigt under utveckling, så om du har en ny utgåva, så kommer där oundvikligen att finnas skillnader. Återigen, detta borde inte skapa några problem, men det skulle kunna ge upphov till viss förvirring.

Det finns två versioner av källkoden till Linux-kärnan, ”production” och ”development” (stabil respektive utvecklings-version. övers.anm.). Production-utgåvor börjar med 1.0.x är för tillfället versionerna med jämna nummer; 1.0.x var production, 1.2.x var production, precis som 2.0.x. Dessa kärnor anses vara de mest stabila, bugg-fria versionerna tillgängliga, då de ges ut. Utvecklings-versionerna av kärnan (1.1.x, 1.3.x osv) är menade som test-kärnor, för folk som har lust att prova på nya och ibland ganska buggiga kärnor. Du har blivit varnad.

### 1.2 Några ord om stilen

`Text som ser ut så här` är antingen något som uppträder på din skärm, ett filnamn, eller något som kan skrivas in direkt, såsom ett kommando, eller parametrar till ett kommando (om du läser detta dokument i rent text-format, så ser det inte annorlunda ut). Kommandon och andra indata citeras ofta (med ‘ ’), vilket skapar följande klassiska kommaterings-problem: om en sådan sak uppträder i slutet av en mening, inom citationstecken, så skriver folk ofta en ‘.’ tillsammans med kommandot, eftersom det amerikanska sättet att citera säger att man ska sätta punkten innanför citationstecknen. Även om det sunda förnuftet (och tyvärr så förutsätter detta att den som har sunt förnuft är van vid den så kallade amerikanska citations-stilen) borde säga åt en att ta bort punkten först, så är det många som helt enkelt inte kommer ihåg det, så jag kommer placera punkten utanför citations-tecknen i sådana fall. Med andra ord, om jag säger år dig att skriva ”`make config`”, så skriver jag ‘`make config`’, inte ‘`make config.`’

### 1.3 Översättarens anmärkningar

Uppdaterade dokumentet 13/11-98, genom att byta ut översättningen av "permissions" till "rättigheter", istället för det sämre "tillåtelser"

## 2 Viktiga frågor och svar på dessa

### 2.1 Vad är det kärnan gör, förresten?

Unix-kärnan fungerar som en medlare mellan dina program och din hårdvara. Först och främst så utför den (eller ser till så att det blir utfört) minneshantering för alla program (processer) som körs, och ser till så att alla får sin rättvisa (eller orättvisa, det beror på hur man ser på saken) tilldelning av processor-cykler. Vidare så tillhandahåller den ett trevligt, ganska portabelt gränssnitt, som gör att programmen kan prata med din hårdvara.

Kärnan gör visserligen mer än så, men dessa grundläggande funktioner är de viktigaste att känna till.

### 2.2 Varför skulle jag vilja uppgradera min kärna?

Nyare kärnor erbjuder generellt möjligheter att prata med flera typer av hårdvara (de har alltså fler enhetsdrivrutiner), de kan ha bättre process-hantering, de kan arbeta snabbare än äldre versioner, de kan vara mer stabila än äldre versioner och de kan rätta till fjantiga buggar i äldre versioner. De flesta uppgraderar sina kärnor för att de vill ha drivrutinerna och bugg-fixarna.

## 2.3 Vilken slags hårdvara stödjer de nyare kärnorna?

Se Hardware-HOWTOn. Eller så kan du ta en titt på '`config.in`'-filen i Linux-källkoden, eller helt enkelt få reda på det när du kör '`make config`'. Detta visar dig all hårdvara som stöds av standard-distributionen av kärnan, men inte allt som Linux stödjer; många vanliga enhets-drivrutiner (såsom PCMCIA-drivrutinerna och band-drivrutinerna) är laddningsbara moduler, vilka utvecklas och distribueras separat.

## 2.4 Vilka versioner av gcc och libc behöver jag?

Linus rekommenderar en viss version av gcc i `README`-filen, som kommer med Linux-källkoden. Om du inte har denna version, så talar gcc-dokumentationen i den rekommenderade versionen av gcc om ifall du behöver uppgradera libc. Detta är inte en svår procedur, men det är viktigt att du följer instruktionerna.

## 2.5 Vad är en laddningsbar modul?

Dessa är delar av kärn-koden, vilka inte länkas (inkluderas) direkt till kärnan. Man kompilerar dem separat, och kan sätta in dem och ta bort dem från kärnan som körs, nästan när som helst. P.g.a. deras flexibilitet, så är de nu det mest populära sättet att koda vissa kärn-funktioner. Många populära enhets-drivrutiner, såsom PCMCIA-drivrutinerna och QIC-80/40 band-drivrutinerna, är laddningsbara moduler.

## 2.6 Hur mycket hårddisk-utrymme behöver jag?

Det beror på just din system-konfiguration. Den komprimerade källkoden är nästan 6 megabytes stor, vid version 2.0.10. De flesta behåller denna även efter uppackningen. Uppackad tar den upp 24 MB. Men det räcker inte med det – du behöver mer för att faktiskt kompilera den. Detta beror på hur mycket stoppar in i din kärna. På en specifik maskin, t.ex., så har jag nätverk, 3com 3C509-drivrutinen och tre filsystem konfigurerade, till vilket det går åt 30 MB. Om vi lägger till den packade Linux-källkoden, så behöver du runt 36 MB för just denna konfiguration. På ett annat system, utan nätverks-enhets-stöd (men fortfarande med nätverks-stöd) och ljudkorts-stöd, tar den upp ännu mera utrymme. En nyare kärna har dessutom med stor säkerhet ett större källkods-träd än en äldre, så rent generellt, om du har en massa hårdvara, så måste du se till att du har en tillräckligt stor hårddisk i röran (och med dagens prisläge kan jag inte låta bli att rekommendera dig att skaffa en till hårddisk, som ett svar på dina utrymmes-problem).

## 2.7 Hur lång tid tar det?

För de flesta är svaret "ganska lång tid". Ditt systems hastighet och mängden minne du har bestämmer, i sista instans, tiden, men det har även en del att göra med mängden grejer du konfigurerar in i kärnan. På en 486DX4/100, med 16 MB RAM, tar det, för en v1.2-kärna med fem filsystem, nätverks-stöd och ljudkorts-drivrutiner runt 20 minuter. På en 386DX/40 (8 MB RAM), med en liknande konfiguration, tar kompileringen nästan en och en halv timme. Det är generellt sett en bra rekommendation att ta en kopp kaffe, se på TV, sticka, eller vad du nu brukar göra för skojs skull, när din maskin kompilerar en kärna. Du kan även be någon med en snabbare maskin kompilera en kärna åt dig, om du har en långsam maskin.

# 3 Hur den faktiska konfigureringen av kärnan går till

## 3.1 Skaffa källkoden

Du kan skaffa källkoden via anonym ftp från `ftp.funet.fi`, i `/pub/Linux/PEOPLE/Linus`, en spegel eller någon annan sajt. Den heter i typfallet `linux-x.y.z.tar.gz`, där `x.y.z` är versionsnumret. Nyare (bättre?) versioner och patcharna finns vanligtvis i underkataloger som '`v1.1`'och '`v1.2`'. Det högsta numret är den senaste versionen, och är vanligtvis en "test-utgåva", vilket innebär att du inte ska ta hem den, om du känner dig osäker på beta- och alfa-utgåvor. Håll dig i så fall till den stabila utgåvan.

Det rekommenderas *starkt* att du använder en spegel-sajt, istället för ftp.funet.fi. Här kommer en kort lista på speglar och andra sajter:

```
USA:        sunsite.unc.edu:/pub/Linux/kernel
USA:        tsx-11.mit.edu:/pub/linux/sources/system
UK:         sunsite.doc.ic.ac.uk:/pub/unix/Linux/sunsite.unc-mirror/kernel
Austria:    ftp.univie.ac.at:/systems/linux/sunsite/kernel
Germany:    ftp.Germany.EU.net:/pub/os/Linux/Local.EUnet/Kernel/Linus
Germany:    sunsite.informatik.rwth-aachen.de:/pub/Linux/PEOPLE/Linus
France:     ftp.ibp.fr:/pub/linux/sources/system/patches
Australia:  sunsite.anu.edu.au:/pub/linux/kernel
```

Rent generellt så är en spegel-sajt till `sunsite.unc.edu` ett bra ställe att leta på. Filen `/pub/Linux/MIRRORS` innehåller en lista på kända speglar. Om du inte har tillgång till ftp, så postas en lista på BBS-system med Linux regelbundet till comp.os.linux.announce; försök få tag på den.

Om du är ute efter generell information om Linux och distributioner, ta en titt på `http://www.linux.org`.

## 3.2 Packa upp källkoden

Logga in som, eller `su`-a till, '`root`', och `cd`-a till `/usr/src`. Om du installerade kärn-källkoden, då du först installerade Linux (som de flesta gör), så finns där redan en katalog som heter '`linux`', vilken innehåller hela källkods-trädet. Om du har gott om hårddisk-utrymme och vill ta det säkra före det osäkra, så behåll den katalogen. En bra idé är att ta reda på vilken version ditt system kör nu och byta namn på katalogen, enligt versions-numret. Kommandot '`uname -r`' skriver ut den aktuella kärnans version. Alltså, om '`uname -r`' säger '`1.0.9`', så bör du ändra namnet (med '`mv`') på '`linux`' till '`linux-1.0.9`'. Om du känner dig hyffsat vårdslös, så kan du helt enkelt ta bort katalogen. I vilket fall som helst, se till så att det inte finns någon '`linux`'-katalog i `/usr/src` innan du packar upp det fullständiga källkods-trädet.

Packa nu upp källkoden i `/usr/src` med '`tar zxpvf linux-x.y.z.tar.gz`' (om du har en `.tar`-fil, utan `.gz` på slutet, så fungerar '`tar xpvf linux-x.y.z.tar`'). Innehållet i källkoden kommer flyga förbi. När uppackningen är klar, kommer det finnas en ny '`linux`'-katalog i `/usr/src`. `cd`-a till `linux` och läs igenom `README`-filen. Där finns en avdelning som heter '`INSTALLING the kernel`'. Utför instruktionerna när det är passande – symboliska länkar som ska finnas på plats, borttagande av onödiga `.o`-filer osv.

## 3.3 Konfigurera kärnan

Observera: En del av detta är upprepningar/klargöranden av en liknande avdelning i Linus `README`-fil.

Kommandot '`make config`', när du är i `/usr/src/linux` startar ett konfigurerings-skal-program, vilket ställer en massa frågor. Det kräver bash, så se till att bash är `/bin/bash`, `/bin/sh` eller `$BASH`.

Det finns några alternativ till '`make config`', och det är mycket möjligt att du kommer tycka att de är lättare och mer bekväma att använda. De som "kör X" kan pröva '`make xconfig`', om de har Tk installerat ('click-o-rama' - Nat). '`make menuconfig`' är för de som har (n)curses och föredrar en text-baserad meny. Dessa gränssnitt har en klar fördel: om du ställer till något dumt, och väljer fel på något alternativ, under konfigureringen, så är det enkelt att gå tillbaks och fixa det.

Du ska vara redo att svara på frågorna, vanligtvis med '`y`' (ja) eller '`n`' (nej). Enhets-drivrutiner har vanligtvis även ett '`m`'-alternativ. Detta betyder "modul", vilket innebär att systemet kommer att kompilera den, men inte direkt in i kärnan, utan som en laddningsbar modul. Ett lustigare sätt att beskriva det är som "maybe" (kanske.övers.anm.). Några av de mer uppenbara och okritiska alternativen beskrivs inte här; se avsnittet "Andra konfigurerings-alternativ" för korta beskrivningar av några andra.

I 2.0.x och sernare finns det ett '?'-alternativ, vilket ger en kort beskrivning av det aktuella alternativet. Den informationen är troligen den mest aktuella.

### 3.3.1 Matte-emulering i kärnan (Math emulation in kernel)

Om du inte har en matte-hjälp-processor (om du har en "bar" 386 eller 486SX), så måste du säga '`y`' till detta. Om du har en hjälp-processor och ändå säger '`y`', så är det inget att oroa sig för – hjälp-processorn används i alla fall, och emuleringen ignoreras. Den enda konsekvens detta får är att kärnan blir större (slöseri med RAM). Jag har blivit informerad om att matte-emuleringen är långsam; även om detta inte har något med detta avsnitt att göra, så kan det vara bra att ha i bakhuvudet om du tycker att X-Window-systemet uppträder klumpigt.

### 3.3.2 Normal (MFM/RLL) disk och IDE disk/cdrom-stöd

Du behöver antagligen stödja detta; det innebär att kärnan kommer att stödja vanliga PC-hårddiskar, vilket de flesta har. Den här drivrutinen inkluderar inte SCSI-hårddiskar; de kommer senare i konfigureringen.

Du kommer sedan bli tillfrågad om "old disk-only"- och "new IDE"-drivrutinerna. Du bör välja en av dessa: den huvudsakliga skillnaden är att den gamla drivrutinen endast stödjer två hårddiskar på ett enda gränssnitt, och att den nya stödjer ett sekundärt gränssnitt och IDE/ATAPI CD-ROM-spelare. Den nya drivrutinen är 4k större än den gamla, och påstås också vara "förbättrad", vilket innebär att, förutom att den innehåller ett annat antal buggar, så kan den förbättra dina hårddiskars prestanda, speciellt om du har nyare (EIDE-typ) hårdvara.

### 3.3.3 Nätverks-stöd (Networking support)

In princip, säg '`y`' endast om din maskin är i ett nätverk, som Internet, eller om du vill använda SLIP, PPP, term osv för uppringd Internet-förbindelse. Men eftersom många paket (som X-Window-systemet) kräver nätverks-stöd, även om maskinen inte finns i ett riktigt nätverk, så bör du säga '`y`'. Senare kommer du bli tillfrågad om du vill stödja TCP/IP-nätverk; säg återigen '`y`' här om du inte är absolut säker.

### 3.3.4 Begränsa minnet till 16 MB (Limit memory to low 16 MB)

Det finns buggiga "386 DMA controllers", vilka har problem med att komma åt mer än 16 MB RAM; säg '`y`' om det skulle råka vara så att du har en.

#### 3.3.5 System V IPC

En av de bästa definitionerna av IPC (Interprocess Communication) finns i Perl book's ordlista. Inte helt överraskande, så använder vissa Perl-programmerare det för att låta processer tala med varandra, och även andra paket (av vilka DOOM är det mest anmärkningsvärda), så det är inte en bra idé att säga `n`, om du inte vet exakt vad du sysslar med.

#### 3.3.6 Processor-typ (386, 486, Pentium, PPro)

(i äldre kärnor: använd -m486-flaggan för 486-specifika optimeringar)

Traditionellt så har detta kompilerat vissa optimeringar för speciella processorer; Kärnorna fungerade fint på andra chips, men kärnan blev kanske en aning större. I nyare kärnor är dock inte detta längre fallet, så du bör välja den processor du kompilerar kärnan för. En "386"-kärna fungerar på alla maskinerna.

#### 3.3.7 SCSI-stöd (SCSI support)

Om du har SCSI-enheter, säg '`y`'. Du kommer bli tillfrågad om vidare information, som t.ex. stöd för CD-ROM, diskar och vilken sorts SCSI-adapter du har. Se SCSI-HOWTOn för mer detaljer.

#### 3.3.8 Stöd för nätverks-enhets (Network device support)

Om du har ett nätverks-kort, eller om du vill använda SLIP, PPP eller en parallell-ports-adapter för att koppla upp dig på Internet, säg '`y`'. Config-programmet kommer fråga efter vilket slags kort du har, och vilket protokoll du använder.

#### 3.3.9 Filsystem

Config-programmet frågar sedan om du vill ha stöd för de följande filsystemen:

Standard (minix) - Nyare distributioner skapar inte minix-filsystem, och många använder det inte, men det kan fortfarande vara en bra idé att ta med detta. Programmen på vissa "räddnings-disketter" använder det, och många disketter kan fortfarande ha minix-filsystemet, eftersom det är mindre smärtsamt att använda på en diskett.

Extended fs - Detta var den första versionen av det utvidgade filsystemet, vilket inte längre används speciellt mycket. Det är antagligen så att du vet om du behöver det och om du är osäker, så behöver du det inte.

Second extended - Det här används mycket i nyare distributioner. Du har antagligen ett sådant, och ska säga '`y`'.

xiafs filesystem - En gång i tiden var det här inte så ovanligt, men när detta skrivs, så vet jag inte om någon som använder det.

msdos - Om du vill använda din MS-DOS-hårddisks partitioner, eller montera MS-DOS-formatterade disketter, säg '`y`'.

umsdos - Det här filsystemet expanderar ett MS-DOS-filsystem med vanliga Unix-liknande funktioner, såsom långa filnamn. Det är inte användbart för folk (som jag), som "inte DOSar".

/proc - Ännu en av de bästa sakerna sedan torrmjölk (idén skamlöst stulen från Bell Labs, antar jag). Man skapar inte ett proc-filsystem på en disk; det här är ett filsystems-gränssnitt till kärnan och processerna. Många process-listare (såsom '`ps`') använder det. Testa '`cat /proc/meminfo`' eller `cat /proc/devices`' ibland. Vissa skal (speciellt rc) använder `/proc/self/id` (känt som `/dev/fd` på andra system) för in/ut-hantering. Du ska antagligen säga '`y`' till det här; många viktiga Linux-verktyg är beroende av det.

NFS - Om din maskin finns i ett nätverk och du vill kunna använda filsystemen, som finns på andra system, med NFS, säg ‘y’.

ISO9660 - Finns på de flesta CD-ROMar. Om du har en CD-ROM-spelare och vill kunna använda den under Linux, säg ‘y’.

OS/2 HPFS - När detta skrivs, ett filsystem som endast tillåter läsning, för OS/2 HPFS.

System V och Coherent - för partitioner i System V- och Coherent-system (det finns andra Unix-varianter för PC).

**Men jag vet inte vilka filsystem jag behöver!** Okej, skriv ‘mount’. Utdatan kommer se ut något i denna stil:

```
blah# mount
/dev/hda1 on / type ext2 (defaults)
/dev/hda3 on /usr type ext2 (defaults)
none on /proc type proc (defaults)
/dev/fd0 on /mnt type msdos (defaults)
```

Titta på varje rad; ordet brevid ‘type’ är filsystems-typen. I det här exemplet, mina /- och /usr-filsystem är ”second extended”, jag använder /proc och det finns en diskett monterad, vilken använder msdos-filsystemet (blä).

Du kan testa ‘cat /proc/filesystem’, om du har /proc på för tillfället; det kommer lista din nuvarande kärnas filsystem.

Att ta med sällan använda, icke nödvändiga filsystem kan leda till att kärnan blir onödigt stor; se avsnittet om moduler för en väg runt detta och ”Fallgropar”-avsnittet, om varför en för stor kärna inte är bra.

### 3.3.10 Tecken-enheter (Character devices)

Här tar du med drivrutiner för din skrivare (parallell-skrivare alltså), buss-mus, PS/2-mus (många bärbara datorer använder PS/2-mus-protokollet för sina inbyggda trackballs), vissa band-spelare (tape drivers) och andra dylika ”tecken”-enheter. Säg ‘y’ där det passar.

Anmärkning: Selection är ett program som låter användaren använda en mus utanför X-Window-systemet, för att kunna klippa och klistra mellan virtuella konsoller. Det är ganska trevligt om du har en seriell mus, för det samexisterar bra med X, men du måste ta till speciella trick för andra. Selection-stöd var ett konfigurerings-alternativ en gång i tiden, men det är nu standard.

Anmärkning 2: Selection anses nu föråldrat. ”gpm” är namnet på det nya programmet. Det kan göra häftigare saker, som att översätta mus-protokoll, hantera mer än en mus...

### 3.3.11 Ljud-kort

Om du känner att du har ett behov av att höra biff skälla, säg ‘y’, och ett annat config-program kommer senare kompilera och fråga efter alla data om ditt ljudkort. (En anmärkning om ljudkorts-inställning: när det frågar dig om du vill installera ”the full version” (den fullständiga versionen) av drivrutinen, så kan du säga ‘n’, och spara en del kärn-utrymme, genom att välja enbart de funktioner du finner nödvändiga.) Jag rekommenderar starkt att du tar en titt på Sound-HOWTO, för mer detaljer om ljud-stöd, om du har ett ljudkort.

#### 3.3.12 Andra konfigurerings-alternativ

Alla konfigurerings-alternativ listas inte här, eftersom de ändras alldeles för ofta, eller är ganska självförklarande (t.ex. 3Com 3c509-stöd, för att kompilera drivrutinen för just detta ethernet-kort). Det finns en ganska innehållsrik lista på alla alternativ (och ett sätta att placera dem i `Configure`-skal-programmet), sammansatt av Axel Boldt (`axel@uni-paderborn.de`), med den följande URLen:

```
http://math-www.uni-paderborn.de/~axel/config_help.html
```

eller via anonym FTP från:

```
ftp://sunsite.unc.edu/pub/Linux/kernel/config/krnl_cnfg_hlp.x.yz.tgz
```

Där `x.yz` är versionsnumret.

För senare (2.0.x och senare) kärnor har detta blivit integrerat i källkods-trädet.

#### 3.3.13 "Kärn-hackande" (Kernel hacking)

>Från Linus README:

"kernel hacking"-konfigurering ger ofta en större eller långsammare (eller båda) kärna, och kan till och med göra kärnan mindre stabil, genom att ställa in vissa rutiner för att aktivt försöka förstöra dålig kod, för att hitta kärn-problem (kmalloc()). Du ska alltså antagligen svara 'n' på den här frågan, om du vill ha en "production"-kärna.

### 3.4 Och nu då? (Makefile)

Efter att du har kört `make config`, kommer ett meddelande som talar om att din kärna har blivit konfigurerad, och att du ska "check the top-level `Makefile` for additional configuration" etc.

Ta alltså en titt på `Makefile`. Du kommer antagligen inte behöva ändra i den, men det skadar inte att ta en titt. Du kan också ändra dess val med '`rdev`'-kommandot, så fort kärnan är på plats.

## 4 Kompilera kärnan

### 4.1 Städa upp och ange beroenden (Cleaning and depending)

När configure-programmet avslutas säger det också åt dig att '`make dep`' och (eventuellt) '`clean`'. Kör alltså '`make dep`'. Detta ser till så att alla beroenden (dependencies), som include-filerna, är på plats. Det tar inte lång stund, om inte din dator är ganska långsam från början. För äldre kärn-versioner ska du köra en '`make clean`' när du är klar. Detta tar bort alla object-filer och några andra saker, som en gammal version lämnar efter sig. I vilket fall som helst, glöm *inte* detta steg, innan du försöker kompilera om en kärna.

### 4.2 Kompilerings-dags

Efter `dep` och `clean`, så kan du nu antingen köra '`make zImage`' eller '`make zdisk` (det är den här delen som tar lång tid). '`make zImage`' kommer kompilera kärnan och lämna en fil i `arch/i386/boot`, vilken heter '`zImage`' (tillsammans med en del andra saker). Detta är den nya, packade kärnan. '`make zdisk`' gör samma sak, men lägger också in den nya `zImage` på en diskett, vilken du förhoppningsvis satt in i floppy-station

"A:". 'zdisk' är ganska praktiskt för att pröva nya kärnor; om den krashar (eller helt enkelt inte fungerar speciellt bra), ta bara ut disketten och starta om med din gamla kärna. Det kan också vara ett praktiskt sätt att starta upp om du av misstag tar bort din kärna (eller något lika hemskt). Du kan också använda den för att installera nya system, när du just har dumpat innehållet från en hårddisk på en annan ("allt det här och mer! HUR mycket skulle du betala?").

Alla åtminstone halvvägs resonabla nya kärnor är packade, därav 'z' framför namnet. En packad kärna packar automatiskt upp sig själv när den körs.

## 4.3 Andra "make"-prylar

'make mrproper' gör en mer omfattande 'clean'. Ibland är det nödvändigt; du kanske vill göra det vid varje patch. 'make mrproper' tar också bort din konfigurerings-fil, så du ska antagligen ta en säkerhetskopia av den (.config) om du tycker den är värdefull.

'make oldconfig' gör ett försök att konfigurera kärnan från en gammal konfigurerings-fil; det kör igenom 'make config'-processen åt dig. Om du aldrig har kompilerat en kärna förut eller inte har en gammal config-fil, så ska du antagligen inte göra detta, eftersom du antagligen kommer ändra standard-konfigureringen.

Se avsnitten om moduler för en beskrivning av 'make modules'.

## 4.4 Installera kärnan

När du har en ny kärna som verkar fungera som du vill att den ska fungera, så är det dags att installera den. De flesta använder LILO (Linux Loader) till detta. 'make zlilo' installerar kärnan, kör LILO på den och fixar allt, så att det bara är att starta upp, MEN ENDAST om lilo är inställd på följande sätt på ditt system: kärnan är /vmlinuz, lilo finns i /sbin och din lilo-konfigurering (/etc/lilo.conf) inte har något att invända.

Annars blir du tvungen att använda LILO direkt. Det är ett ganska enkelt paket att installera och arbeta med, men det har en tendens att förvirra människor med sin konfigurerings-fil. Titta på konfigurerings-filen (antingen /etc/lilo/config, för äldre versioner, eller /etc/lilo.conf, för nyare versioner), och kolla upp den nuvarande konfigureringen. lilo.conf-filen ser ut så här:

```
image = /vmlinuz
    label = Linux
    root = /dev/hda1
    ...
```

'image =' anger den kärna du för tillfället har installerad. De flesta använder /vmlinuz. 'label' används av lilo för att bestämma vilken kärna, eller vilket operativ-system, som ska startas, och 'root' är / för just det operativ-systemet. Gör en säkerhets-kopia av din gamla kärna och kopiera zImage-filen, vilken du just skapat, dit den ska vara (du skulle köra 'cp zImage /vmlinuz' om du använder '/vmlinuz'). Kör sedan lilo igen – på nyare system så kan du köra 'lilo', men på äldre grejer kanske du blir tvungen att köra /etc/lilo/install, eller till och med en /etc/lilo/lilo -C /etc/lilo/config.

Om du skulle vilja veta mer om LILOs konfigurering, eller om du inte har LILO, skaffa den senaste versionen från din favorit-ftp-sajt och följ instruktionerna.

För att starta upp en av dina gamla kärnor från hårddisken (ett annat sätt att rädda dig om du gör något dumt med den nya kärnan), så kopiera raderna nedan (och inklusive) 'image = xxx' i LILOs konfigurerings-fil till slutet av filen, och modifiera 'image = xxx' till 'image = yyy', där 'yyy' är den fullständiga sökvängen till filen du sparade din säkerhetskopia av kärnan som. Ändra sedan 'label = zzz' till 'label = linux-backup' och kör `lilo` igen. Det kan vara nödvändigt att lägga in en rad som säger '`delay=x`', där x är antalet tiondelar, vilket säger åt LILO att vänta så lång tid innan det startar upp, så att du kan avbryta det (med shift-knappen t.ex.), och skriva in den säkerhets-kopierade kärnans "label" (om något otrevligt skulle hända).

# 5 Patcha kärnan

## 5.1 Lägga till en patch

Gradvisa uppgraderingar till kärnan distribueras som patchar. Om du t.ex. har version 1.1.45 och du upptäcker att det finns en '`patch46.gz`' till den, så innebär det att du kan uppgradera till version 1.1.46 genom att lägga till patchen. Det kan vara bra att ta en säkerhets-kopia av källkods-trädet först ('`make clean`' och sedan '`cd /usr/src; tar zcvf old-tree.tar.gz linux`', vilket kommer att skapa ett komprimerat tar-arkiv åt dig).

Om vi fortsätter på exemplet ovan, låt oss förutsätta att du har '`patch46.gz`' i `/usr/src`. `cd`-a till `/usr/src` och kör en '`zcat patch46.gz | patch -p0`' (eller '`patch -p0 < patch46`', om patchen inte är packad). Du kommer se saker flyga förbi (eller krypa förbi, om ditt system är så långsamt), vilka talar om för dig att det försöker lägga till bitar, och huruvida det lyckas eller ej. Vanligtvis går det här för fort för att du ska hinna med att läsa, och du kan inte vara så säker på om det fungerade eller ej, men om du använder `-s`-flaggan till `patch` så kommer `patch` endast rapportera fel (du får inte lika mycket av den där "hallå, min dator gör faktiskt saker, för en gångs skull!"-känslan, men du kanske föredrar det här...). För att leta efter delar som inte har gått så bra, cd-a till `/usr/src/linux` och leta efter filer med en `.rej`-ändelse. Vissa versioner av `patch` (äldre versioner, vilka kan har kompilerats på ett underlägset filsystem) lämnar dessa misslyckanden med en `#`-ändelse. Du kan använda '`find`' för att leta;

```
find .  -name '*.rej' -print
```

skriver ut, till standard ut, vilka filer, i den aktuella katalogen eller underkataloger, somhar `.rej`-ändelser

Om något gick fel, gör en '`make clean`', '`config`' och '`dep`', som det beskrivs i avsnitt 3 och 4.

Det finns en hel del alternativ till `patch`-kommandot. Som nämnts ovan så kommer `patch -s` att hålla tyst om allt utom fel. Om du har ditt kärn-källkods-träd någon annanstans än i `/usr/src/linux`, så patchar `patch -p1` allt korrekt (i den katalogen). Andra `patch`-alternativ finns väl-dokumenterade på man-sidan.

## 5.2 Om något går snett

(Observera: detta avsnitt hänvisar mestadels till ganska gamla kärnor)

Det vanligaste problemet som brukade uppstå, var när en patch modifierade en fil som heter '`config.in`', och den inte såg riktigt rätt ut, eftersom du ändrade alternativen för att passa din maskin. Det här har man tagit hand om, men man kan fortfarande stöta på det hos äldre utgåvor. För att fixa det, ta en titt på `config.in.rej`-filen, och se efter vad som finns kvar av original-patchen. Förändringarna är vanligtvis markerade med '`+`' och '`-`' i början av raderna. Ta en titt på raderna före och efter, och tänk efter om de var satta till '`y`' eller '`n`'. Editera nu `config.in`, och ändra '`y`' till '`n`' och '`n`' till '`y`', där det ska vara så. Gör en

```
patch -p0 < config.in.rej
```

och om det rapporteras att det lyckades (inga fel), så kan du fortsätta med konfigurering och kompilering. `config.in.rej` kommer finnas kvar, men du kan ta bort den.

Om du stöter på vidare problem, så kan du ha lagt till en patch i fel ordning. Om patch säger '`previously applied patch detected: Assume -R?`', så försöker du antagligen lägga till en patch som är under ditt aktuella versionsnummer; om du svarar '`y`', så kommer patch försöka nedgradera din källkod, och kommer antagligen misslyckas; då kommer du alltså behöva skaffa ett helt nytt källkodsträd (vilket kanske inte skulle ha varit en så dum idé från början).

För att backa ut (ta bort) en patch, använd '`patch -R`' på original-patchen.

Det bästa du kan göra när patchningar verkligen går snett är att börja om igen, med ett rent och opatchat källkodsträd (från en av `linux-x.y.z.tar.gz`-filerna, t.ex.) och börja om.

### 5.3 Bli av med .orig-filer

Efter bara några få patchar kommer `.orig`-filerna vara ganska många. T.ex. så hade ett 1.1.51-källkodsträd jag hade blivit rensat vid 1.1.48. Att ta bort .orig-filerna sparade mig mer än en halv meg.

```
find .  -name '*.orig' -exec rm -f {} ';'
```

tar hand om det åt dig. De versioner av `patch` som använder `#` för misslyckaden, använder en tilde istället för `.orig`.

Det finns bättre sätt att bli av med `.orig`-filerna, vilka är baserade på GNU `xargs`:

```
find .  -name '*.orig' | xargs rm
```

eller "ganska säker men en aning mer pratig"-metoden:

```
find . -name '*.orig' -print0 | xargs --null rm --
```

### 5.4 Andra patchar

Det finns andra patchar (jag kallar dem "icke-standard") än de som Linus distribuerar. Om du lägger till dessa så kanske inte Linus patchar fungerar korrekt, och du blir tvungen att antagligen backa ut, fixa källkoden eller patchen, installera ett nytt källkodsträd eller en kombination av dessa saker. Detta kan bli väldigt frustrerande, så om du inte vill modifiera källkoden (med möjligheten av väldigt dåliga resultat), ta bort icke-standard-patcharna innan du lägger till Linus, eller installera helt enkelt ett nytt träd. Sen kan du se efter om icke-standard-patcharna fortfarande fungerar. Om de inte gör det, så är du antingen fast med en gammal kärna, och blir tvungen att greja en massa med patcharna eller källkoden, för att få det att fungera, eller tvingas vänta på (eller möjligtvis be om) en ny version av patchen.

Hur vanliga är patcharna, som inte ingår i standard-distributionen? Du kommer antagligen höra talas om dem. Jag brukade använda noblink-patchen för min virtuella konsoll, eftersom jag hatar blinkande markörer (denna patch blir (eller blev, åtminstone) uppdaterad för varje ny utgåva av kärnan). Eftersom de flesta nya enhets-drivrutiner utvecklas som laddningsbara moduler, så minskar dock antalet icke-standard-patchar en hel del.

## 6 Ytterligare paket

Din Linux-kärna har många funktioner som inte förklaras i själva källkoden; dessa funktioner används i typfallet av externa paket. Några av de vanligaste räknas upp här.

### 6.1 kbd

Linux-konsollen har fler funktioner än den förtjänar. Bland dessa finns möjligheten att byta typsnitt, definiera om tangentbordet, byta grafik-lägen (i nyare kärnor) osv. kbd-paketet har program som låter användaren göra allt detta, plus många typsnitt och tangentbords-tabeller för i stort sett alla tangentbord, och är tillgängligt från samma sajter som har kärn-källkoden.

### 6.2 util-linux

Rik Faith (`faith@cs.unc.edu`) plockade ihop en stor samling Linux-verktyg vilka, av ett underligt sammanträffande, kallas util-linux. Dessa utvecklas nu av Nicolai Langfeldt (`util-linux@math.uio.no`). Det är tillgängligt via anonym ftp från sunsite.unc.edu, i `/pub/Linux/system/misc`, och innehåller program såsom `setterm`, `rdev` och `ctrlaltdel`, vilka är relevanta för kärnan. Som Rik säger, *installera inte utan att tänka efter*; du behöver inte installera allt i paketet, och det kan mycket väl skapa stora problem, om du gör det.

### 6.3 hdparm

Som med många andra paket, så var detta en gång i tiden en kärn-patch och stöd-program. Patcharna blev inplockade i den officiella kärnan, och programmen för att optimera och leka med hårddisken distribueras separat.

### 6.4 gpm

gpm står för general purpose mouse. Detta program tillåter dig att klippa och klistra text mellan virtuella konsoller och göra andra saker, med en stor uppsättning olika mus-typer.

## 7 Några fallgropar

### 7.1 make clean

Om din nya kärna gör riktigt knasiga saker, efter en rutin-uppgradering av kärnan, så finns det en möjlighet att du glömt att `make clean`, innan du kompilerade den nya kärnan. Symptomen kan vara allt från att ditt system helt enkelt kraschar och konstiga in/ut-problem, till dålig prestanda. Se till att du gör en `make dep` också.

### 7.2 Enorma eller långsamma kärnor

Om din kärna suger åt sig en massa minne, är för stor och/eller bara tar en evighet att kompilera, även när du har fått din nya 786DX6/440 att fungera med den, så har du antagligen konfigurerat in en massa onödiga saker (drivrutiner, filsystem osv.). Det du inte använder ska du inte ta med, för det tar upp minne. De mest uppenbara symptomen på för stora kärnor är extrem swappning till och från hårddisken; om din hårddisk för en massa oväsen, och den inte är en sådan där gammal Fujitsu Eagle, som lät som en jumbo-jet som var på väg att landa när du slog av dem, så ta en titt på din kärn-konfigurering.

Du kan ta reda på hur mycket minne kärnan använder genom att ta den totala mängden minne i din maskin och subtrahera summan av ”total mem” i `/proc/meminfo`, eller utdatan från kommandot ‘`free`’. Du kan också ta reda på det genom att köra ‘`dmesg`’ (eller genom att titta på kärnans log-fil, var den nu finns på ditt system). Där finns en rad som ser ut någonting i stil med detta:

```
Memory: 15124k/16384k available (552k kernel code, 384k reserved, 324k data)
```

Min 386 (vilken har en aning mindre skräp in-konfigurerat) säger:

```
Memory: 7000k/8192k available (496k kernel code, 384k reserved, 312k data)
```

Om du bara 'måste' ha en stor kärna, men systemet inte låter dig ha det, så kan du testa 'make bzimage'. Du kan mycket väl bli tvungen att installera en ny version av LILO för att få detta att fungera.

## 7.3 Kärnan kompilerar inte

Om den inte kompilerar så är det troligt att en patch misslyckades, eller att din källkod är korrupt på något sätt. Det kan också vara något fel på din version av gcc, eller den kan vara korrupt (det kan t.ex. vara något fel på include-filerna). Se till att de symboliska länkar, som Linus beskriver i README-filen ser ut som de ska. Rent generellt kan man säga, att om en standard-kärna inte kompilerar, så är det något stort fel på ditt system, och en ominstallering av vissa saker är antagligen nödvändig.

Eller du kanske kompilerar en 1.2.x-kärna med en ELF-kompilator (gcc 2.6.3 eller högre). Om du får en hög det-och-det undefined-meddelanden under kompileringen, så finns det vissa möjligheter att det är detta som är ditt problem. Lösningen är i de flesta fall väldigt enkel. Lägg till följande rader i början av arch/i386/Makefile:

```
AS=/usr/i486-linuxaout/bin/as
LD=/usr/i486-linuxaout/bin/ld -m i386linux
CC=gcc -b i486-linuxaout -D__KERNEL__ -I$(TOPDIR)/include
```

Kör sedan make dep och zImage igen.

I sällsynta fall kan gcc krascha p.g.a. hårdvaru-problem. Fel-meddelandet kommer vara något i stil med "xxx exited with signal 15", och det kommer oftast se mystiskt ut. Jag skulle antagligen inte nämna detta, om det inte hade hänt mig en gång - jag hade dåligt cache-minne, och kompilatorn kunde begå slumpmässiga fel. Pröva först med att ominstallera gcc, om du får problem. Du bör endast börja bli misstänksam om din kärna kompilerar bra med "external cache" avslaget, en mindre mängd RAM osv.

Det brukar irritera folk, när det sägs att det kan vara något problem med deras hårdvara. Tja, jag sitter inte och hittar på det här. Det finns en FAQ om detta – den finns på http://www.bitwizard.nl/sig11/.

## 7.4 Den nya versionen av kärnan verkar inte starta (boot)

Du körde inte LILO, eller konfigurerade det inte korrekt. En sak som "fick" mig en gång, var ett problem med konfigurerings-filen; den sa 'boot = /dev/hda1', istället för 'boot = /dev/hda'. (Detta kan först vara väldigt irriterande, men så fort du har en fungerande konfigurerings-fil så behöver du inte göra några ändringar i den.)

## 7.5 Du glömde köra LILO, eller systemet startar inte alls

Hoppsan! Det bästa du kan göra här är att starta upp med en diskett och preparera en till start-diskett (som 'make zdisk' skulle göra). Du måste veta var ditt rot-filsystem (/) finns, och vilken sort det är (t.ex. second extended, minix). I exemplet nedan måste du också veta vilket filsystem ditt /usr/src/linux-källkodsträd finns på, dess typ och var det vanligtvis monteras.

I det följande exemplet är / /dev/hda1, och filsystemet som innehåller /usr/src/linux är /dev/hda3, och monteras vanligtvis på /usr. Båda filsystemen är second extended. Den fungerande kärnan i /usr/src/linux/arch/i386/boot heter zImage.

Idén är, att om det finns en fungerande `zImage`, så är det möjligt att använda den till en ny diskett. Ett annat alternativ, vilket kan fungera bättre eller sämre (det beror på metoden du använde när du stökade till i filsystemet), diskuteras efter detta exempel.

Först och främst, starta från en start/rot-diskett-kombination eller en räddnings-diskett och montera filsystemet vilket innehåller den fungerande kärnan:

```
mkdir /mnt
mount -t ext2 /dev/hda3 /mnt
```

Om `mkdir` säger åt dig att katalogen redan existerar, ignorera det bara. `cd`-a nu till stället där den fungerande kärnan finns. Observera att

```
/mnt + /usr/src/linux/arch/i386/boot - /usr = /mnt/src/linux/arch/i386/boot
```

Sätt i en formatterad diskett i diskett-station ”A:” (inte din start- eller rot-diskett!), dumpa kärnan på disketten och konfigurera den för ditt rot-filsystem:

```
cd /mnt/src/linux/arch/i386/boot
dd if=zImage of=/dev/fd0
rdev /dev/fd0 /dev/hda1
```

`cd`-a till `/` och avmontera det vanliga `/usr`-filsystemet:

```
cd /
umount /mnt
```

Du ska nu kunna starta om ditt system som vanligt från denna diskett. Glöm inte att köra lilo (eller vad det nu var som gick snett) efter att du startat om!

Som nämndes ovan, så finns det ett annat vanligt alternativ. Om du råkade ha en fungerande kärna i `/` (`/vmlinuz` t.ex.), så kan du använda den till en start-diskett. Om vi förutsätter alla de ovannämnda omständigheterna, och att kärnan är `/vmlinuz`, gör bara följande ändringar i exemplet ovan: ändra `/dev/hda3` till `/dev/hda1` (/-filsystemet), `/mnt/src/linux` till `/mnt`, och `if=zImage` till `if=vmlinuz`. Anmärkningen som förklarar hur du tar reda på `/mnt/src/linux` kan du hoppa över.

Att använda LILO med stora hårddiskar (mer än 1024 cylindrar) kan leda till problem. See LILO mini-HOWTO eller dokumentationen för hjälp om det.

## 7.6 Jag får ‘warning: bdflush not running’

Det här kan vara ett allvarligt problem. Från och med en kärn-utgåva efter 1.0 (runt 20 april 1994), uppgraderas/utbyttes ett program som heter ‘`update`’, vilket regelbundet spolar ur (flushes) filsystemets buffrar. Skaffa källkoden till ‘`bdflush`’ (du kan hitta det där du hittade källkoden till kärnan), och installera det (det är bäst att köra systemet med den gamla kärnan, då du gör detta). Det installerar sig självt som ‘`update`’, och efter att du startat om ska den nya kärnan inte längre klaga.

## 7.7 Den säger saker om ”undefined symbols” och kompilerar inte

Du har antagligen en ELF-kompilator (gcc 2.6.3 eller senare) och källkoden till kärna 1.2.x (eller tidigare). Den vanliga lösningen är att lägga till följande tre rader i början av `arch/i386/Makefile`:

```
AS=/usr/i486-linuxaout/bin/as
LD=/usr/i486-linuxaout/bin/ld -m i386linux
CC=gcc -b i486-linuxaout -D__KERNEL__ -I$(TOPDIR)/include
```

Detta kommer kompilera en 1.2.x-kärna med a.out-bibliotek.

## 7.8 Jag får inte min IDE/ATAPI CD-ROM-spelare att fungera

Konstigt nog så är det många som inte kan få sina ATAPI-spelare att fungera, antagligen för att det finns ett antal saker som kan gå snett.

Om din CD-ROM-spelalre är den enda enheten på ett visst IDE-gränssnitt så måste de jumpras som ”master” eller ”single”. Detta påstås vara det vanligaste felet.

Creative Labs (som ett exempel) har satt IDE-gränssnitt på sina ljudkort nu. Detta leder dock till det intressanta problemet att, medan vissa bara har ett gränssnitt från början, så har många IDE-gränssnitt inbyggda i sina moderkort (vanligen på IRQ15), så en vanlig praxis är att göra soundblaster-gränssnittet till en tredje IDE-port (IRQ11, åtminstone är det vad jag har hört).

Detta skapar problem med Linux, genom att version 1.2.x inte stödjer ett tredje IDE-gränssnitt (stödet dyker upp någonstans i 1.3.x-serien, men det är en utvecklings-serie och den auto-söker inte). För att komma runt detta, kan du välja mellan några olika metoder.

Om du redan har en andra IDE-port, så finns det risk att du inte använder den, eller att den redan inte har två enheter. Ta ATAPI-spelaren från ljudkortet och sätt det på det andra gränssnittet. Sedan kan du stänga av ljudkortets gränssnitt, vilket i alla fall sparar in en IRQ.

Om du inte har ett andra gränssnitt, jumpra ljudkortets gränssnitt (inte ljudkortets ljud-delar) som IRQ15, det andra gränssnittet. Detta bör fungera.

Om den, av någon anledning, absolut måste vara på ett så kallat ”tredje” gränssnitt, eller om det finns andra problem, skaffa en 1.3.x-kärna (1.3.57 har det, t.ex.) och läs igenom `drivers/block/README.ide`. Det finns mycket mer information där.

## 7.9 Den säger konstiga saker om ”obsolete routing requests”

Skaffa nya versioner av `route`-programmet och alla andra program som sysslar med route-manipulering. `/usr/include/linux/route.h` (vilken faktiskt är en fil i `/usr/src/linux`) har ändrats.

## 7.10 Brandvägg fungerar inte i 1.2.0

Uppgradera till åtminstone version 1.2.1.

## 7.11 “Not a compressed kernel Image file”

Använd inte `vmlinux`-filen, vilken skapas i `/usr/src/linux` som kärna; `[..]/arch/i386/boot/zImage` är den rätta.

## 7.12 Problem med konsoll-terminalen efter uppgradering till 1.3.x

Ändra ordet `dumb` till `linux` i konsollens termcap-rad i `/etc/termcap`. Du kan också bli tvungen att skapa en terminfo-avdelning.

### 7.13 Verkar inte som om jag kan kompilera saker efter kärn-uppgradering

Källkoden till Linux-kärnan innehåller ett antal include-filer (sakerna som slutar med `.h`), vilka hänvisas till av standard-versionerna i `/usr/include`. De hänvisas vanligtvis till på detta sätt (där `xyzzy.h` skulle vara något i `/usr/include/linux`):

```
#include <linux/xyzzy.h>
```

Vanligtvis finns det en länk, som heter `linux` i `/usr/include` till `include/linux`-katalogen i din kärn-källkod (`/usr/src/linux/include/linux` i ett vanligt system). Om denna länk inte finns där, eller pekar till fel ställe, så kommer det mesta att vägra kompilera. Om du bestämmer dig för att källkoden till kärnan tar upp för mycket plats på hårddisken, och tar bort den, så är detta uppenbarligen ett problem. Ett annat sätt det kan gå fel på är med fil-rättigheter; om din `root` har en umask, som inte tillåter andra användare att se dess filer som standard, och du packade upp kärn-källkoden med `p`-parametern (bevara fil-lägen), så kommer de andra användarna inte att kunna använda C-kompilatorn. Även om du skulle kunna använda `chmod`-kommandot för att fixa detta, så är det antagligen enklare att packa upp include-filerna igen. Du kan göra detta på samma sätt som du packade upp hela källkoden från början, men med en ytterligare parameter:

```
blah# tar zxvpf linux.x.y.z.tar.gz linux/include
```

Observera: ”`make config`” kommer återskapa `/usr/src/linux`-länken, om den inte finns där.

### 7.14 Utöka gränser

De följande få *exempel*-kommandona kan vara hjälpsamma för de som undrar hur man kan utöka vissa ”mjuka” gränser, som sätts av kärnan:

```
echo 4096 > /proc/sys/kernel/file-max
echo 12288 > /proc/sys/kernel/inode-max
echo 300 400 500 > /proc/sys/vm/freepages
```

## 8 Anmärkning om att uppgradera till version 2.0.x

Version 2.0.x av kärnan introducerade en hel del förändringar, vad gäller installering av kärnan. Filen `Documentation/Changes` i källkodsträdet till 2.0.x innehåller information om saker du bör veta, när du uppgraderar till version 2.0.x. Du kommer antagligen behöva uppgradera flera viktiga paket, såsom gcc, libc och SysVInit, och kanske modifiera vissa system-filer, så räkna med detta. Ta det bara lugnt.

## 9 Moduler

Laddningsbara kärn-moduler kan spara minne och göra konfigureringen enklare. Modulernas omfattning har växt till att inkludera filsystem, drivrutiner till ethernet-kort, bandspelare, drivrutiner till skrivare m.m.

### 9.1 Installera modul-verktygen

Modul-verktygen är tillgängliga från samma ställe som du hittar källkoden till kärnan, som `modules-x.y.z.tar.gz`; välj den högsta patch-nivån `x.y.z`, som är lika med eller under den till din aktuella kärna. Packa upp det med ‘`tar zxvf modules-x.y.z.tar.gz`’, `cd`-a till katalogen det skapar (modules-x.y.z), ta en titt på README-filen och utför det du instrueras att göra (vilket vanligtvis är något enkelt, som make install). Du ska nu ha programmen insmod, rmmod, ksyms, lsmod, genksyms, modprobe och depmod i /sbin. Om du vill så kan du testa verktygen med "hw"-exempel-drivrutinen i insmod; titta till INSTALL-filen i den underkatalogen, för detaljer.

insmod infogar en modul i en kärna som körs. Moduler har vanligtvis en .o-ändelse; exempel-drivrutinen, som nämns ovan, heter drv_hello.o, så för att infoga denna skriver du 'insmod drv_hello.o'. För att ta reda på vilka moduler kärnan använder för tillfället, använd lsmod. Utdatan ser ut så här:

```
blah# lsmod
Module:        #pages:  Used by:
drv_hello         1
```

'drv_hello' är namnet på modulen, det använder en minnes-sida (page) (4k), och inga andra kärn-moduler är beroende av den för tillfället. För att ta bort den här modulen, använd 'rmmod drv_hello'. Observera att rmmod vill ha ett *modul-namn*, inte ett filnamn; du får reda på detta från lsmods uppräkning. De andra modul-verktygen dokumenteras i sina man-sidor.

## 9.2 Moduler som distribueras med kärnan

I version 2.0.30 är nästan allt tillgängligt som laddningsbara moduler. För att använda dem måste du först se till att inte konfigurera in dem i den vanliga kärnan; säga alltså inte y till dem under 'make config'. Kompilera en ny kärna och starta om med den. cd-a sedan till /usr/src/linux igen och kör en 'make modules'. Detta kompilerar alla de moduler som du inte specificerade i kärn-konfigureringen, och placerar länkar till i /usr/src/linux/modules. Du kan använda dem direkt från den katalogen, eller köra 'make modules_install', vilket installerar dem i /lib/modules/x.y.z, där x.y.z är versionen av kärnan.

Detta kan vara särskilt praktiskt med filsystem. Du kanske inte använder minix- eller msdos-filsystemen så ofta. Om jag t.ex. stötte på en msdos-diskett (usch då), så skulle jag insmod /usr/src/linux/modules/msdos.o, och sedan rmmod msdos när jag var klar. Denna procedur spara runt 50k RAM i kärnan, under normal användning. En liten anmärkning är på sin plats angående minix-filsystemet: du ska *alltid* konfigurera in det direkt i kärnan, för användning på "räddnings"-disketter.

# 10 Andra konfigurerings-alternativ

Detta avsnitt innehåller beskrivningar av utvalda konfigurerings-alternativ (i make config), vilka inte tas upp i konfigurerings-avsnittet. De flesta enhets-drivrutiner tas inte upp här.

## 10.1 Vanliga inställningar (General setup)

Normal floppy disk support - är precis det (stöd för vanliga floppy-diskar). Du kan ta en titt på filen drivers/block/README.fd; detta är extra viktigt för IBM Thinkpad-användare.

XT harddisk support - om du vill använda den där 8-bitars XT-controllern, som ligger i hörnet och samlar damm.

PCI bios support - om du har PCI, så kan du pröva på det här; ta det försiktigt, bara, eftersom vissa gamla PCI-moderkort kommer krascha av det här. Mer information om PCI-bussen under Linux finns i PCI-HOWTOn.

Kernel support for ELF binaries - ELF är ett försök att låta binär-filer fungera på olika arkitekturer och operativ-system; Linux verkar vara på väg i den riktningen, så du vill antagligen ha detta.

`Set version information on all symbols for modules` - förut kompilerades alltid kärn-modulerna om med varje ny kärna. Om du säger `y` så kommer det bli möjligt att använda moduler som kompilerats under en annan patch-nivå. Läs `README.modules` för mer detaljer.

### 10.2 Networking options

Networking options (nätverks-alternativ) beskriv i NET-3-HOWTOn (eller NET-någonting-HOWTOn).

## 11 Tips och knep

### 11.1 Omdirigera utdatan från make- eller patch-kommandona

Om du skulle vilja ha log-filer, om vad de där '`make`'- eller '`patch`'-kommandona gjorde, så kan du omdirigera utdatan till en fil. Ta först reda på vilket skal du kör: '`grep root /etc/passwd`' och leta efter någonting i stil med '`/bin/csh`'.

Om du använder sh eller bash, kör

```
(kommando) 2>&1 | tee (output file)
```

vilket placerar en kopia av `(kommando)`s utdata i filen '`(output file)`'.

För csh eller tcsh, använd

```
(command) |& tee (output file)
```

För rc (Observera: du använder antagligen inte rc) är det

```
(command) >[2=1] | tee (output file)
```

### 11.2 Villkorlig kärn-installering

Förutom att använda disketter, så finns det ett flertal metoder för att pröva en ny kärna, utan att röra den gamla. I motsats till flera andra Unix-varianter så kan LILO starta en kärna från var som helst på hårddisken (om du har en stor (500 MB eller mer) hårddisk, läs igenom LILO-dokumentationen, om hur detta kan leda till problem). Alltså, om du lägger till något i stil med

```
image = /usr/src/linux/arch/i386/boot/zImage
    label = new_kernel
```

till slutet av LILOs konfigurerings-fil, så kan du välja att köra en nyligen kompilerad kärna, utan att röra din gamla `/vmlinuz` (efter att du kört `lilo` naturligtvis). Det enklaste sättet att säga åt LILO att starta en ny kärna att trycka ner shift-tangenten då systemet startas (när det står `LILO`, och inget annat, på skärmen), vilket ger dig en prompt. Nu kan du skriva '`new_kernel`' för att starta den nya kärnan.

Om du vill ha flera olika kärn-källkodsträd på ditt system, på samma gång (detta kan ta upp en *massa* hårddisk-utrymme; var försiktig), så är det vanligaste sättet att kalla dem `/usr/src/linux-x.y.z`, där `x.y.z` är kärn-versionen. Du kan sedan "välja" ett källkodsträd med en symbolisk länk; t.ex. så skulle '`ln -sf linux-1.2.2 /usr/src/linux`' göra 1.2.2 till det aktuella trädet. Innan du skapar en sådan symbolisk länk, se till så att det sista parametern till `ln` inte är en riktig katalog (gamla symboliska länkar går bra); resultatet kanske inte blir vad du väntat dig.

### 11.3 Uppdateringar av kärnan

Russell Nelson (`nelson@crynwr.com`) sammanfattar förändringarna i nya utgåvor av kärnan. Dessa är korta och du kan ta en titt på dem innan du utför en uppgradering. De är tillgängliga via anonym ftp från `ftp.emlist.com` i `pub/kchanges`, eller genom URLen

```
http://www.crynwr.com/kchanges
```

## 12 Andra relevanta HOWTOn, som kan vara användbara

- Sound-HOWTO: ljud-kort och -verktyg
- SCSI-HOWTO: allt om SCSI-controllers och -verktyg
- NET-2-HOWTO: nätverk
- PPP-HOWTO: PPP-nätverk
- PCMCIA-HOWTO: om drivrutinerna till din bärbara dator
- ELF-HOWTO: ELF: vad det är, konvertering...
- Hardware-HOWTO: översikt av stödd hårdvara
- Module-HOWTO: mer om kärn-moduler
- Kerneld mini-HOWTO: om kerneld
- BogoMips mini-HOWTO: ifall du undrade

## 13 Diverse

### 13.1 Författare

Författaren och utvecklaren av Linux Kernel-HOWTO är Brian Ward (`bri@blah.math.tu-graz.ac.at`). Var vänlig skicka mig kommentarer, tillägg, rättelser (rättelser är speciellt viktiga för mig).

Du kan ta en titt på min 'hemsida' på en av dessa URLer:

```
http://www.math.psu.edu/ward/
http://blah.math.tu-graz.ac.at/~bri/
```

Även om jag försöker ta så mycket hänsyn som möjligt till e-post, var vänlig kom ihåg att jag får en *massa* e-post varje dag, så det kan ta ett tag att svara. Speciellt om du skickar mig en fråga, försök extra mycket att vara klar och detaljerad i ditt meddelande. Ifall du inte skriver om icke fungerande hårdvara (eller något åt det hållet), så behöver jag veta hur din hårdvaru-konfigurering ser ut. Om du rapporterar ett fel, säg inte bara "Jag försökte det här men fick ett fel". Jag behöver veta vad det var för fel. Jag skulle också vilja veta vilka versioner av kärnan, gcc och libc du använder. Om du bara säger att du använder den-eller-den distributionen, så säger det mig inte så mycket. Jag bryr mig inte om att du frågar enkla frågor; kom ihåg att, om du inte frågar, så får du aldrig reda på svaret! Jag vill tacka alla som har gett mig respons.

Om du har skickat mig ett e-brev och inte fått något svar inom rimlig tid (tre veckor eller så), så finns det en möjlighet att jag av misstag har raderat ditt meddelande eller något (ursäkta mig). Försök igen.

Jag får en massa e-brev om saker som faktiskt är hårdvaruproblem. Det är okej, men försök komma ihåg att jag inte känner till all hårdvara i hela världen och att jag inte vet hur hjälpsam jag kan vara; personligen använder jag en maskin med IDE- och SCSI-hårddiskar, SCSI CD-ROM, 3Com och WD ethernet-kort, seriella möss, moderkort med PCI, NCR 810 SCSI-controllers, AMD 386DX40 med Cyric hjälp-processor, AMD5x86, AMD 486DX4 och Intel 486DX4-processorer (detta är en överblick över vad jag använder och känner till, definitivt inte en rekommendation, men om det är vad du vill ha så är du mer än välkommen att fråga `:-)`).

Version -0.1 skrevs 3 oktober 1994. Detta dokument är tillgängligt som SGML, PostScript, TeX, roff och ren text.

## 13.2 Att göra

"Tips och knep"-avsnittet är lite litet. Jag hoppas kunna utöka det med förslag från andra.

Samma med "Ytterligare paket".

Mer information om hur man klarar av avlusning/krash-återhämtning behövs.

## 13.3 Bidrag

En liten del av Linux README (kärn-hackning-alternativet) är inkluderad. (Tack Linus!)

`uc@brian.lunetix.de` (Ulrich Callmeier): patch -s and xargs.

`quinlan@yggdrasil.com` (Daniel Quinlan): rättelser och tillägg i många avsnitt.

`nat@nat@nataa.fr.eu.org` (Nat Makarevitch): mrproper, tar -p, många fler saker

`boldt@math.ucsb.edu` (Axel Boldt): samlade beskrivningar av konfigurerings-alternativ på nätet; gav mig sedan listan

`lembark@wrkhors.psyber.com` (Steve Lembark): flera uppstarts-förslag (boot suggestions) `kbriggs@earwax.pd.uwa.edu.au` (Keith Briggs): några rättelser och förslag

`rmcguire@freenet.columbus.oh.us` (Ryan McGuire): make-tillägg

`dumas@excalibur.ibp.fr` (Eric Dumas): fransk översättning

`simazaki@ab11.yamanashi.ac.jp` (Yasutada Shimazaki): japansk översättning

`jjamor@lml.ls.fi.upm.es` (Juan Jose Amor Iglesias): spansk översättning

`mva@sbbs.se` (Martin Wahlen): svensk översättning

`jzp1218@stud.u-szeged.hu` (Zoltan Vamosi): ungersk översättning

`bart@mat.uni.torun.pl` (Bartosz Maruszewski): polsk översättning

`donahue@tiber.nist.gov` (Michael J Donahue): skrivfel, vinnare i "skivat bröd-tävlingen"

`rms@gnu.ai.mit.edu` (Richard Stallman): dokumentation om "fri" dokumentation och kommentarer om detta

`dak@Pool.Informatik.RWTH-Aachen.DE` (David Kastrup): NFS-saker

`esr@snark.thyrsus.com` (Eric Raymond): diverse småsaker

De som har skickat mig e-brev med frågor och problem har också varit ganska hjälpsamma.

## 13.4 Upphovsrätt, licens och sådana grejer



Rekommendationer: Kommersiell vidaredistribuering är tillåten och uppmuntras; det rekommenderas dock starkt att distributören tar kontakt med författaren innan vidaredistribueringen, för att kunna hålla saker och ting aktuella (du kan gärna skicka mig ett exemplar av saker du tillverkar när du ändå håller på). Översättare rekommenderas också att ta kontakt med författaren innan de översätter. Den tryckta versionen ser snyggare ut. Återanvänd.