# TTK4235: Intro til Linux

Kolbjørn Austreng

Vår 2020

## Beskrivelse

Denne oppgaven er ment som en rask innføring til Linux, slik at resten av labopplegget fremover ikke foregår i et helt ukjent miljø. Sannsynligvis har mange av dere har erfaring med Linux fra før - og det er på ingen måte et krav at dere følger denne oppgaven til punkt og prikke om dere allerede er komfortable med Linux.

Den eneste seksjonen dere "må" gå gjennom er seksjon 2. Resten av oppgaveteksten er "unødvendig" i den forstand at dere ikke trenger kunnskapen fra dem for å bruke Linux - men den er inkludert allikevel for de av dere som måtte ønske litt bakgrunn.

# 1 Historie

Om lag en mannsalder tilbake var datamaskiner store og komplekse beist uten sidestykke. Hver datamaskin bestod av særdeles særegen hardware, og kjørte særdeles særegen software. Alle programmer var skreddersydd rundt én spesifikk arkitektur, uten mulighet for å kjøre på en hvilken som helst annen plattform. Nye systemer krevde stort sett alltid opplæring fra bunnen av, ettersom veldig lite var overførbart mellom forskjellige maskiner.

I 1969 påbegynte Bell Labs utviklingen av et system som skulle løse mange av kompatibilitetsproblemene datidens datamaskiner hadde. Istedenfor å skrive det nye systemet i assembler - som hittil var normen - utviklet de et nytt språk for det nye systemet; C. Selve systemet ble hetende "UNIX".

En av de banebrytende trekkene ved UNIX var fokus på kodegjenbruk. Tidligere operativsystemer var alle skrevet for å støtte en gitt type hardware. UNIX separerte ut denne hardwarespesifikke koden i en samling kalt en *kernel*, og implementerte resten av den høynivå funksjonaliteten rundt kernelen istedenfor rundt hardwaret direkte. Resultatet var at kun kernelen måtte endres fra plattform til plattform, mens resten av operativsystemet kunne forbli det samme.

UNIX hadde noen gode år mens nye leverandører av hardware og softwareutviklere la til støtte for nye plattformer. Allikevel var UNIX stort sett forbeholdt universiteter og større selskaper. Rundt slutten av 80-tallet var personlige datamaskiner mer utbredt, men UNIX var den gang ganske uegnet med tanke på hvor drepende tregt det kjørte på hardwaret man kunne finne i hjemmedatamaskiner. Rundt samme tid ble MS DOS og Windows 3.1 populært, nettopp fordi det var forholdsvis raskt - etter datidens standarder.

I 1991 fant en Finsk datastudent ved navn Linus Torvalds ut at han ville lage en fri klone av UNIX - fri i betydning av åpen kilde og uten restriksjonene ved vanlig UNIX.

En av grunnene til at det lot seg gjøre å skrive en UNIX-klone er at UNIX allerede da var spesifisert via POSIX - eller *Portable Operating System Interface*. POSIX er en IEEE standard som spesifiserer hvordan APIet til operativsystemet skal se ut, og hvordan enkelte verktøy skal fungere.

Linux i seg selv implementerer derimot ikke alle de verktøyene man trenger for å ha et brukbart operativsystem. Eksempelvis er det veldig greit å kunne kompilere kode, redigere filer, lese epost, etc. For alt av dette bruker Linux et sett med verktøyer utgitt av et annet prosjekt kalt GNU. Av den grunn refereres Linux av og til som "GNU/Linux" - men dette er stort sett im-

plisitt når man nevner Linux.

I dag får man Linux via et utall forskjellige distribusjoner som alle gjør ting ganske likt - men fortsatt forskjellig nok til at alle kan finne noe de liker hvis de prøver lenge nok. Ulempen er selvsagt at det er vanskelig å finne "riktig" distribusjon på første forsøk. På Sanntidssalen brukes den populære varianten Ubuntu.

# 2 Monkey Level Basics

Dette er den viktigste seksjonen i oppgaven, og dekker stort sett det du trenger for å bruke Linux i de kommende labene. Ubuntu er distribusjonen som er installert på Sanntidssalen, men alt dette gjelder for enhver variant av Linux-baserte operativsystemer.

Ubuntu kommer med et vennlig grafisk grensesnitt som lar deg bruke maskinen på "vanlig" måte. Det er helt greit å gjøre det, men det stort sett raskere å gjøre ting via en terminal når bruken er kjent. Du kan åpne en terminal ved å trykke **Ctrl + Alt + T**, eller ved å åpne "Dash" (øverst til venstre) og søke etter "terminal".

Terminalen vil åpne i hjemmemappen din, og prompten vil gjerne se slik ut (avhengig av hvem som har gjort endringer på maskinen før deg, kan den være annerledes):

```
student@Ubuntu:~$
```

Tildesymbolet (∼) forteller deg at du er i din egen hjemmemappe, og $ forteller deg at du er en vanlig bruker - i motsetning til *root* (en allmektig bruker), som symboliseres ved "#".

## 2.1 pwd

Om prompten ikke viser deg hvor du er, har du også kommandoen `pwd`, som forteller deg hvilken mappe du befinner deg i - `/home/student`.

## 2.2 ls

For å vise innholdet i mappen du befinner deg i, har du kommandoen `ls`. Om du kaller `ls` fra hjemmemappen vil du typisk se en rekke mapper:

```
Documents Downloads Pictures Music
Public Videos Desktop Templates
```

Dette er de samme mappene du vil se om du åpner Ubuntu sin filutforsker; som du kan åpne ved å trykke ikonet til venstre, eller søke "nautilus" i dash.

Du kan også se innholdet i en mappe ved å kalle `ls mappenavn`.

Som mange andre kommandoer, har `ls` en rekke *flagg* du kan legge til for å endre oppførselen til kommandoen. For eksempel vil flagget `-l` liste opp hvilke typer filer som finnes, hvem som kan bruke dem, hvem som eier dem, filstørrelse, sist modifikasjonsdato, og navn:

```
$ ls -l

total 488
drwxr-xr-x  2 student student   4096 Nov 18 01:49 Documents
drwxr-xr-x  2 student student   4096 Nov 18 01:49 Downloads
drwxr-xr-x  2 student student   4096 Nov 18 01:49 Pictures
drwxr-xr-x  2 student student   4096 Nov 18 01:49 Music
drwxr-xr-x  2 student student   4096 Nov 18 01:49 Public
drwxr-xr-x  2 student student   4096 Nov 18 01:49 Videos
drwxr-xr-x  2 student student   4096 Nov 18 01:49 Desktop
drwxr-xr-x  2 student student   4096 Nov 18 01:49 Templates
```

Formatet som `ls -l` spytter ut, er illustrert i figur 1. Den første bokstaven ($d$) betyr at filen er en mappe[1]. Deretter følger noen bokstaver som forteller hvem som kan gjøre hva med filen. En $r$ betyr at du har leserettighet, en $w$ betyr at du har skriverettighet, og en $x$ betyr at du kan kjøre filen.

I eksempelet i figur 1 kan eieren av filen gjøre hva han eller hun vil, mens filens gruppe og alle andre kan gjøre alt bortsett fra å endre på filen.

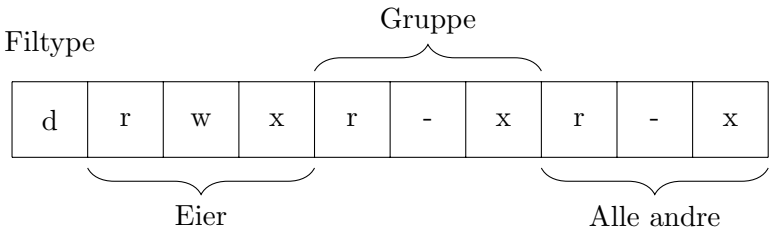


Figure 1: Filegenskaper fra `ls -l`.

Etter denne filbeskrivelsen kommer et magisk tall (2 i eksempelet). Dette tallet er antall *hardlenker* filen har, og har med hvordan filen er lagret på. Dette antallet er det helt greit å ignorere.

Deretter følger navn på eieren av filen, *student*, og gruppen filen tilhører;

[1] I Linux er mapper også betraktet som filer.

også *student*. Tallet 4096 som følger hver av linjene i figur 1 er antall byte som filen okkuperer - men ettersom mapper egentlig bare er "pekere" som forteller hvilke filer den inneholder, er dette tallet faktisk størrelsen på "pekeren" - altså ikke størrelsen på det mappen inneholder.

Til slutt kommer en timestamp av når filen sist ble endret, og navnet på filen.

### 2.3 cd

For å navigere inn- og ut fra mapper brukes `cd mappenavn`. Eksempelvis kaller du `cd Downloads` om du ønsker å bevege deg inn i mappen "Downloads".

For å bevege deg et nivå opp bruker du to punktum som mappenavnet; `cd ..` vil altså ta deg tilbake til hjemmemappen hvis du er i "Downloads".

I Linux betyr to punktum alltid "min foreldremappe", mens ett punktum betyr "mappen jeg er i". Det betyr også at kommandoen `cd .` vil flytte deg inn i mappen du allerede befinner deg i.

Om du har beveget deg inn i en avsidesliggende mappe, for eksempel ved å kalle `cd /etc/systemd/network`, kan du komme deg tilbake til dit du sist var ved å kalle `cd -`.

### 2.4 file

Om du lurer på hva en fil er for noe, kan du bruke kommandoen `file filnavn`. For eksempel:

```
$ file Downloads
Downloads: directory

$ file main.c
main.c: C source, ASCII text
```

### 2.5 cat

For å raskt ta innholdet i en fil og skrive det ut i terminalen har du kommandoen `cat filnavn`:

```
$ cat main.c
#include <stdio.h>

int main(){
```

```
        printf("Hello world\n");
        return 0;
}
```

### 2.6 man

Når du lurer på hva en kommando gjør - og vil være mer *pro hacker* enn å sjekke StackOverflow - har du `man`, kort for *manual page*.

Om du lurer på hvilke flagg `ls` støtter, kan du kalle `man ls`, for å få manualen til `ls`.

Om du lurer på hvordan du bruker `man`, kan du også kalle `man man`. Om dere gjør det, vil dere se at `man` grupperer manualene i 9 kategorier - hvor kategori 3 er for bibliotekkall.
Om du trenger en spesifikk kategori, kan du også spesifisere det fra kommandolinjen. For eksempel har du programmet `printf`, som du vil få ved å kalle `man printf`. Om du derimot vil ha dokumentasjonen på funksjonen i C, kan du kalle `man 3 printf`.

### 2.7 sudo

Enkelte kommandoer er forbeholdt brukere med ekstra rettigheter. Om du får *permission denied* når du forsøker å kalle en kommando, kan du midlertidig eskalere dine egne rettigheter ved å kalle den med `sudo` foran (for **s**uper**u**ser **do**).

### 2.8 apt

Denne kommandoen er spesifikk til Ubuntu og andre Linuxvarianter som bruker `apt` pakkeverktøyet.

De fleste Linuxdistribusjoner har en veldig elegant måte å håndtere installerte programmer på - nemlig gjennom en pakkemanager. Dette lar deg installere og oppdatere den programvaren du vil, uten å måtte loke rundt på et utall nettsider for å finne en installasjonsfil som blir utdatert innen en måned. For eksempel kan du installere programmeringsspråket Ruby sin interpreter ved å kalle `sudo apt install ruby`. Informasjon om hva pakken `ruby` inneholder kan fås ved å kalle `apt show ruby`.

For å oppdatere alle installerte pakker til nyeste versjon som Ubuntu har tilgjengelig, kaller dere `sudo apt update`, etterfulgt av `sudo apt upgrade`. For informasjon om hvilke kommandoer `apt` støtter, kan dere som ellers kalle `man apt`.

En ting som er verdt å vite om er at Ubuntu gjerne liker å teste nye pakker en stund før de legges til i oversikten som `apt` bruker i utgangspunktet. Med andre ord kan det ta en stund før nyeste versjon av programvare kommer til Ubuntu.

## 2.9 Oppgaver

1. Naviger til rotmappen (`cd /`), og bruk `ls` og `cd` for å komme deg tilbake til din egen hjemmemappe uten å bruke `cd -`.

2. I Linux er det mange filer som starter med et punktum i navnet. Disse er stort sett ikke vist i filutforskeren eller av `ls` med mindre du ber om det. Bruk `man ls` til å finne hvilket flagg du kan gi `ls` for å vise alle filene i en mappe.

3. Linux kan "pipe" output fra en kommando inn i en annen. Et eksempel på det er `cat main.c | less`, som vil ta utskriften av "main.c" og fôre inn i kommandoen `less`. Prøv dette på en lang fil. Dere kan forøvrig bruke `man less` for å finne ut hvordan `less` fungerer.

4. Filutforskerprogrammet som kommer med Ubuntu heter "nautilus", og kan startes fra terminalen ved å kalle `nautilus`. Prøv å kall `nautilus . &` (altså med punktum som argument). &-tegnet er for at terminalen skal starte nautilus i bakgrunnen istedenfor å henge når vinduet er oppe.

5. Oppgrader alle pakkene på systemet.

# 3 Filsystemet

Å ha en inngående kjennskap til filsystemet i Linux er ikke nødvendig for å kunne bruke det, men det er fortsatt greit å vite om med tanke på at Linux har en litt annen filosofi enn Windows når det kommer til filer:

> “ On a UNIX system, everything is a file; if something is not a file, it is a process. ”

En mappe er simpelthen en fil som peker til andre filer; programmer, bilder, eller musikkfiler er egentlig bare tekst som tolkes på en annen måte; hardware kan beskrives med egne spesielle filer, etc.

## 3.1 Typer filer

Når alt er en fil, er det greit å ha en måte å skille dem fra hverandre på. Kommandoen `ls` har de følgene måtene å skille filer på:

| Type | Forklaring | Type | Forklaring |
|---|---|---|---|
| - | Vanlig fil | b | Blokkenhet |
| c | ”Character”-fil | C | Høyhastighetsfil |
| d | Mappe | D | ”Door” (Solaris) |
| l | Lenke | M | Migrert fil |
| n | Nettverksfil | p | Navngitt FIFO *pipe* |
| P | Port | s | Socket |
| ? | Ukjent fil | | |

De fleste av disse typene er litt esoteriske; dere kommer desidert oftest borti vanlige filer (-) og mapper (d). Om dere skulle finne filer av andre typer vet dere nå i det minste hva dere skal søke etter på StackOverflow.

## 3.2 Filsystemets struktur

Filsystemet på Linux kan sees på som et tre. Roten av treet er mappen ”/”, som dere kan ta en titt på ved å kalle `ls /`. Filene som finnes i `/` kan variere noe, men stort sett forventer vi å finne disse:

| Undermappe | Funksjon |
|---|---|
| bin | Felles programmer, delt av brukere og systemet |
| boot | Oppstartsfiler og kernelen, ofte også *bootloader*data |
| dev | ”Devices”; perifere enheter tilkoblet systemet |

| | |
|---|---|
| etc | De fleste viktige konfigurasjonsparametrene til systemet |
| home | Mappen med alle brukere sine hjemmemapper. Deres hjemmemappe ligger under `/home/student` |
| initrd | "Initial RAM Disc"; viktig oppstartsfil |
| lib, lib64 | Biblioteker; ting dere kan `#include` |
| lost+found | Stedet hvor filer som blir reddet av under diskfeil ligger |
| misc | Diverse uklassifiserte filer |
| mnt | "Standard" tilkoblingspunkt for eksterne harddisker etc |
| net | Standard tilkoblingspunkt for eksterne filsystemer |
| opt | Typisk sted for valgfri- og tredjepartsprogramvare |
| proc | Et virtuelt filsystem med informasjon om systemressurser (`man 5 proc`) |
| root | Hjemmemappen til *root user*. Mappen "/" kalles også "root", og det er stort sett den som menes |
| sbin | Programmer med ekstra privilegier brukt av systemet |
| tmp | Midlertidige filer, tømmes ved reboot |
| usr | Programmer, biblioteker, og dokumentasjon for brukerrelaterte programmer |
| var | Alle variable filer, som automatiske logger |

Det er ikke meningen å pugge hva alle undermappene er til, og det er uansett informasjon dere kan få tak i ved å kalle `man 7 hier`. Poenget er bare å illustrere hvordan filer i Linux stort sett er gruppert.

For en bruker er det nok å akseptere at dette filsystemets utseende. I realiteten ligger ikke filene lagret som et tre, men som en stor tabell av *inoder* (index node). Hver inode forteller operativsystemet hvem som eier filen, hvilken type fil det er, når den ble laget, størrelse, hvor den ligger lagret, og en del ting til. Dette er ikke noe dere trenger å kjenne til for å bruke filer i Linux, men for de spesielt interesserte av dere, kan dere se filer sine

inodenummer ved å kalle `ls -i`.

## 3.3 Hvor kjøres programmer fra?

Alle kommandoer, som `ls` og `cd`, er programmer dere kjører hver gang dere kaller dem fra terminalen. For å eksempelvis sjekke hvor `ls` kjører fra kan dere kalle `whereis ls`. Typisk output vil være noe i duren av "`ls: /usr/bin`".

Grunnen til at Linux finner programmet `ls` er at det ligger i en *miljøvariabel* kalt `PATH`. Denne variabelen inneholder en liste av mapper som søkes gjennom hver gang dere forsøker å kalle et program. Om programmet finnes i `PATH`, vil det kjøres. Hvis det ikke finnes i `PATH` vil dere typisk få noe som "`bash: finnes_ikke: command not found`".

Dere kan liste opp alle filene i `PATH` ved å kalle `echo $PATH | tr : '\n'`. Vanlig output ser omtrent slik ut:

```
/usr/local/bin
/usr/local/sbin
/usr/bin
/usr/bin/site_perl
/usr/bin/vendor_perl
/usr/bin/core_perl
```

Om et program finnes i flere av mappene, er det den første varianten av programmet som blir funnet som kalles.

Når dere senere i faget skal kompilere egne programmer og kjøre dem, vil mappen dere er i som regel ikke ligge i `PATH`. Dere kan da fortelle Linux at dere ønsker å kjøre fra en bestemt mappe ved å bruke absolutt eller relativ sti. Som et eksempel kan vi ta "Hello world" i C:

### 3.3.1 Hello World

Lag en ny mappe ved å kalle `mkdir mappenavn`, og naviger inn i den ved å kalle `cd mappenavn`. Om dere nå kaller `ls` vil dere se at den nye mappen er tom.

Dere kan nå lage filen "main.c" ved å åpne den i en editor av deres eget valg. Dere kan for eksempel bruke Gedit ved å kalle `gedit main.c`. Deretter putter dere inn følgende kode og lagrer.

```c
#include <stdio.h>

int main(){
```

```
        printf("Hello world!\n");
        return 0;
}
```

Den mest brukte C-kompilatoren i Linux er GCC (GNU Compiler Collection). Dere kan kompilere fra terminalen ved å kalle `gcc main.c`. Dette vil produsere filen "a.out" (assembler out). Hvis dere nå kaller `ls -l` vil se noe tilsvarende dette:

```
-rwxr-xr-x 1 student student 16568 Jan 27 21:25 a.out
-rw-r--r-- 1 student student    72 Jan 27 21:25 main.c
```

Som dere ser fra *eæcute*-flagget, er `a.out` en kjørbar fil, men om dere prøver å kalle `a.out` fra terminalen, vil Linux klage på at kommandoen ikke finnes. Dette er fordi den ikke ligger i `PATH`-variabelen. Det er her absolutte og relative stier kommer inn:

Dere kan kalle `a.out` ved å spesifisere hvor den ligger, for eksempel ved å kalle `/home/student/mappenavn/a.out`. Dette kalles en absolutt sti fordi den starter med rotmappen (/) og spesifiserer nøyaktig hvor i filsystemet `a.out` ligger.
Alternativt kan dere simpelthen kalle `./a.out` - hvor "`.`" betyr "denne mappen". Dette kalles en relativ sti, fordi "`.`" alltid vil bety forskjellige ting, avhenging av hvor dere er i filsystemet.

Når dere er ferdige med å leke, kan dere rydde opp etter dere ved å kalle `cd ..` (flytt ett mappenivå opp), etterfulgt av `rm -rf mappenavn` (remove *recursive*, *force*). Til slutt kan dere kalle `man rm` for å lære om `rm` :)

# 4 Prosesser

Mange kommandoer, som for eksempel `ls`, starter én eneste *prosess*, som utfører en jobb og avslutter. Det trenger derimot ikke være slik for alle kommandoer eller programmer dere kommer over; enkelte programmer kan starte flere prosesser på en gang - gjerne i bakgrunnen. I tillegg til dette har det tradisjonelt vært vanlig (på UNIX' tid) å ha flere brukere per datamaskin, gjerne logget inn samtidig. Siden alle brukere som er på systemet til enhver tid skal kunne kalle vilkårlige kommandoer, er det viktig at Linux er i stand til å håndtere mange prosesser på en gang.

## 4.1 Interaktive prosesser

Enkelte kommandoer er interaktive; de krever en bruker for å starte (i motsetning til å startes automatisk av systemet), og de okkuperer terminalen så

lenge de kjører. Eventuelt kan interaktive prosesser av og til startes i bakgrunnen, gjerne som et eget vindu, slik at terminalen fortsatt kan akseptere kommandoer.

Et eksempel på en interaktiv prosess er kommandoen `python`, som vil starte en Python-interpreter i terminalen. Terminalen vil være opptatt så lenge Python kjører.

Enkelte programmer kan også startes i bakgrunnen. Dette gjøres ved å legge til et "&"-tegn. Det er dette vi gjorde tidligere, da vi kalte `nautilus . &` for å starte Ubuntu sin filutforsker.

## 4.2 Automatiske prosesser

Linux kjører alltid en rekke ekstra prosesser uten at en bruker trenger å starte dem. Et eksempel på prosesser som kjører uten en bruker er ting som startes på en tidsbasis, gjerne via programmet `cron` eller `at`.

En annen gruppe prosesser, kalt *daemons*, er også veldig viktig for Linux. Daemonprosesser er lokale servere som kjører kontinuerlig. Et eksempel på en slik prosess er `PostgreSQL`, som er en veldig populær relasjonell database. Om `PostgreSQL` er installert på systemet, vil den typisk stare en prosess ved maskinoppstart, som dere senere kan kommunisere med for å behandle informasjonen lagret i databasen(e).

For å ta en titt på hvilke daemoner som kjører på systemet deres, kan dere kalle `systemctl`. Dere vil da få opp en oversikt over alle aktive og inaktive daemoner, og status over dem. Trykk `q` for å komme ut av oversikten.

## 4.3 Signaler

Alle prosesser på Linux påvirkes av *signaler*. Disse signalene kan gjøre ting som å be en prosess avbryte det den gjør, eller avslutte kjøringen. For eksempel vil signalet `SIGINT` (signal interrupt) sendes til en interaktiv prosess om dere trykke `Ctrl + C`. Prosesser står fritt til å ignorere dette signalet - som eksempelvis `python` gjør om dere prøver.

Et sterkere signal, `SIGKILL` (signal kill), kan tvinge en prosess til å avslutte uansett hva den gjør. Dere kan sende signaler til prosesser ved å bruke kommandoen `kill`, men det er stort sett bedre å avslutte prosesser på vanlig vis. For å lære mer om signaler kan dere kalle `man 7 signal`.

# 5 Hva nå?

Linux er et ganske logisk og elegant operativsystem, men det er ingen grunn til å drukne seg i detaljer før dere trenger dem. Om dere aldri har sett en terminal før nå, er det ingen grunn til panikk - ved å gradvis eksponere seg til flere konsepter og verktøy utenfor sin egen komfortsone lærer man mye.

I gamle dager hadde Linux noen aldri så små "RTFM-vibber" rundt seg, men i dag er det stort sett hjelp å få på StackOverflow, AskUbuntu, Arch Linux Wiki, og andre steder - så det er ingen grunn til å stange hode om du sitter fast.

## 5.1 Installere Linux

Dere kan installere Linux på egen maskin i tillegg til å bruke maskinene på Sanntidssalen. Dette trenger ikke bytte ut operativsystemet dere har nå - en ordning kalt *dual booting* er veldig populær. Da har dere to eller flere operativsystemer installert på én datamaskin, så velger dere hvilket dere vil boote inn i ved oppstart.

Installasjonsprosessen er litt forskjellig fra maskin til maskin, og Linux-variant, men her er en rask beskrivelse av en vanlig prosess:

### 5.1.1 Last ned operativsystemet

Det første du trenger er et *image* av operativsystemet du vil installere. Om du aldri har vært borti Linux før, er Ubuntu et greit sted å starte. Fra **ubuntu.com** kan du laste ned nyeste LTS-utgave (long term support). Dette kommer som en `iso`fil som vi trenger å legge over på et oppstartbart medium som en minnepenn.

### 5.1.2 Kopier isofilen til et oppstartsmedium

De fleste datamaskiner har USB-porter som de kan starte fra, så en minnepenn er ideelt for dette. Om du kommer fra Windows eller Mac kan du bruke programmer som `UNetbootin`. Om du har tilgang på en annen Linuxmaskin kan du bruke kommandoen `dd` for å gjøre kopieringen:

1. Koble minnepennen i datamaskinen og kall `lsblk` for å se hva den ble registrert som - typisk som "`sdb`".
2. Kall `dd if=~/Downloads/ubuntu[...]amd64.iso of=/dev/sdb bs=4M status=progress`. Om isofilen ligger et annet sted enn i Downloads endrer dere på `if`-stien. Tilsvarende endrer dere på `of`-stien om minnepennen ble registrert som noe annet enn `sdb`.

### 5.1.3 Start fra oppstartsmediet

Etter at du har laget en oppstartbar minnepenn, kan du starte datamaskinen fra denne. Hvordan dette gjøres, varierer sterkt, men ofte kommer du inn i maskinens *boot meny* ved å trykke `F12`, `F8`, eller `F2` mens maskinen holder på å starte.

På nyere utgaver av Windows kan maskinen også ha *UEFI-beskyttelse*, som først må skrus av, før du kan starte fra et medium som ikke først er godkjent av Microsoft. Linux er *ikke* godkjent av Microsoft; det ville vært dårlig for kundebasen over tid.

### 5.1.4 Next, next, finish

Når du først får startet fra minnepennen, er det bare å prøve ut systemet som du vil, eller installere ved å følge wizarden som kommer opp.