Den Store AV-Bog

Søren Pilgård - DIKUrevyen

13. maj 2017

Indhold (kort)

Indhold

Note: TODO: Skal det omtales som Master, Controller eller kontrolmaskine? Skal det kaldes en kommando eller en instruktion?

Note: De gamle stationærere kan overhovedet ikke følge med til krævende opgaver, videoer og afspiller musik med dårlig kvalitet

Note: Husk at teste lyd. Kablet er sært, man kan risikere at den ikke registrerer det eller noget. Altså at kablet er sat i, men der ikke kommer lyd over det. TEST DETTE, HVAD ER DET HELT KONKRET DER FEJLER???

1 Introduktion

"Er der en AV-mand tilstede?!?"

Sådan kunne spørgsmålet lyde. Når du har gjort dig bekendt med denne bog er du forhåbentligt istand til at svare *"Ja!"*. Denne bog henvender sig til dig der aldrig har lavet AV i en revy før og skal have en udførlig introduktion, til dig der har lavet AV før men lige skal have lidt hjælp til hvordan det nu var med det hersens AV-system, og til dig der har lavet AV i mange år men stadig har behov for at slå op i dokumentationen i ny og næ.

Dette er en introduktion til DIKUrevyens **AV-system**. Systemet er udviklet af Søren Pilgård, og selv om det startede hos DIKUrevyen har det allerede haft bred udbredelse blandt andet i DIKUrevyen, SaTyRrevyen, Biorevyen og Matematikrevyen. Systemet er af den slags der nok aldrig bliver helt færdigt, der er altid lige noget mere der kan tilføjes, laves om og forbedres, det samme gælder denne bog. Det bør dog ikke afskrække dig fra at kaste dig ud i det.

Dette kapitel samt kapitel 3 forklarer lidt generelt om hvad det vil sige at lave revy og hvad AV-manden laver, samt nogle guidelines til hvordan man gør det godt som AV-mand.

At vi kalder det AV-*mand* skal du ikke tage så nøje, der findes super seje og kreative AV-folk af alle køn!

1.1 Hvordan foregår en revy?

Dette er måske din første revy, eller måske bare din første revy på Natfak. Her vil vi for god ordens skyld kort gennemgå hvad en revy i denne sammenhæng indebærer. De fleste revyer har en nogenlunde ens opbygning dette gælder især for revyerne i SaTyRfællesskabet som indefatter DIKURevyen, Fysikrevyen, Matematikrevyen, Biorevyen, MBKrevyen, og så selvfølgeligt SaTyRrevyen selv. En revy er overordnet delt op i forskellige numre, f.eks. sange eller sketches, disse numrer er, som regel, men bestemt ikke altid uafhængige af hinanden uden overlap i hverken karakterer, tema eller indhold. Der sker dog ofte undtagelser iform af f.eks. følgetoner med fortløbene handling, eller f.eks. sketches der naturligt glider over i en sang. Numrene er grupperet i akter af en halv time til tre kvarters varighed med en 15-20 minutters pause. Der er typisk 2 eller 3 akter efterfulgt af et par ekstranumre.

En klassik revy foregår således at klokken 19.00 åbner dørene til foyeren, typisk er de allerede åbne, men det er her folk begynder at stille sig i kø uden foran dørene til auditoriet. Kl. 19.15 åbner dørene ind til auditoriet og publikum strømmer ind. Når folk har fundet deres pladser sker det ofte at de forskellige studieretninger begynder at synge af hinanden. Dette er en god gammel tradition hvor nye svarvers dukker op og stemningen bliver sat. Det er dårlig stil når revyen går i gang mens folk synger, så man skal sørge for at time det før eller efter en sang. Når klokken er 19.30 og backstage er klar signalerer en boss eller instruktør til TEXnikken at nu kan vi starte. Når der ikke synges sætter lysmanden så revyen i gang ved at dæmpe lyset i salen. Et øredøvende sus vil da gå igennem auditoriet som hele det feststemte publikum fylder med råb, banken og hujen.

Nu er revyen i gang!

De fleste revyer starter med en bandintro, en video der præsenterer bandet, når videoen er færdig sætter bandet sig til rette på bandscenen mens folk brøler *"Bandet!, Bandet!..."* nogle revyer har så en velkomst sketch hvor man præsenterer revyen på en sjov måde (fysik starter f.eks. altid med at aflyse revyen), andre går bare direkte over i en startsang der skal få gang i salen.

Nogle revyer foretrækker at holde 20 minutters pause, men skrive 15

Herefter følger en masse blandede sange og sketches i form af akt 1. Man slutter typisk en akt med en sang med gang i så folk går feststemte til pause. Det er lysmandens opgave at åbne op for lyset, men AV manden starter typisk med at have et pauseskilt/billede/video til at fortælle at nu er der pause først. Herefter er der pause i 15-20 minutter mens publikum køber øl og går på toilettet. (Som AV-mand kan det også være en god ide at benytte toiletterne nede backstage og fylde op på vand/drikkevarer nu). Lidt før pausen er klar begynder bandet at spille ind, de spiller typisk noget musik der passer til deres tema eller som de har brugt mens de øvede sig.

Så starter næste akt (hver opmærksom på at der altid er et par stykker der kommer forsent og som konsekvent glemmer at lukke dørene (så hav en løber klar)). I en 3 akters forestilling kører andet akt stille og roligt (sådan konstruktionsmæsigt), i en 2 akters svarer det til at gå direkte til 3. Hver opmærksom på at nogle 2 akters forestillinger kan have en tendens til at have lidt længere akter. Sidste akt slutter af med et slutnummer, en god sang der får folk op og køre. 3/4 inde i sangen kommer alle skuespillerne ud og der siges tak til alle der skal have tak og alle på scenen bukker et par gange. Så lunter folk ud af scenen og lyset går ned så der bliver helt mørkt. Imens klapper folk ihærdigt og råber *"Ekstra nummer!, Ekstra nummer!..."* for det siger traditionen. Det er ofte enten AV eller lys der starter ekstra numrene man skal altid trække den lidt, så trække den lidt mere, og så liiiige trække den lidt mer', både fordi sådan er det, samt for at de kan nå at skifte kostumer/gøre rekvisiter klar. Og så går ekstranumrene i gang. Disse består som regel af 3-4 numre, nogle gange en video og slutter af med en slut-slutsang. Denne sidste sang er handler ofte om at nu er revyen slut, men der er efterfest bagefter og så skal den have fuld gas. Så er revyen færdig, man kan smide et skilt op om at der er efterfest. Hvis man har flere forestillinger men kun én efterfest kan man smide en henvisning om at gå på *Caféen?!* op.

Generelt prøver man ofte at holde det intelligente humor i første akt hvor folk stadig kan forstå det og ikke ville "kede" sig, mens 3 akt er mere plat humor, dårlige ordspil og billige punchlines. I takt med at publikum bliver mere og mere fulde passer dette også meget fint til hvad de magter. Publikum kan være helt fantastiske, men de kan også være larmende, forstyrende og ret overvældende. Dette gælder både for dem

på scenen, men det kan også være skræmmende når TEXnikken laver en fejl og hele auditoriet brøler *"TeXnikken sejler"*. Det vigtigste er at holde hovedet koldt i en hver situation. Få ro på, tænk igennem hvad der går galt, lad være med at opildne publikum, fokuser, og få tingene tilbage på sporet. Vi har **ALLE** været derude hvor det hele gik galt, og bagefter gik det hele altsammen, bare husk på ovenstående.

Hvis du er i tvivl om et begræb så tag et kig i kapitel 1.3.

1.2 Hvad laver en AV-mand?

Hvad er det så AV-manden laver til en revy? Jo, ser du, TeXnikken består af 3 dele, der er lyd, de sørger for at bandets musik, sangernes sang og skuespillernes grynten kommer ud af højtalerne. Lydmanden laver så at sige ikke noget lyd selv, han sørger bare for at alle kan høre det og at det lyder godt. Lysmanden sørger for at man kan se noget, det gør han med et væld af farvede lamper og spots, med lys og skygge samt en gang taktisk røg kan han transformere stemningen og skabe en fest på scenen.

Nogle revyer er også begyndt at have en Epelipsiadvarsel, i starten af revyen og før blinkende numre

AV-mandens rolle er mere alsidig, en slags digital rekvisit man kan bruge til lidt af hvert. Det kunne f.eks. være at agere powerpoint-show til en "fremvisning", vise billeder til at illustrere forskellige pointer, vise videoer, vise pauseskilte, vise hvilke sponsorer der er, afspille digital musik til dansenumre, afspille lydeffekter som pistolskud og prutter. En af de vigtigste opgaver er dog at lave overtekster. Det kan være utroligt svært at lave god lyd i Store UP1, hvis sangerne samtidigt er lidt usikre og der er en masse larm i auditoriet kan det være utroligt svært at høre hvad der bliver sunget. Samtidigt er der mange sange hvor man gerne vil give publikum mulighed for at synge med i omkvædende, derfor har man valgt at indføre overtekster. Overtekster er ligesom undertekster på en film, de sungne tekster bliver bare vist på et hvidt lærred oppe bag scenen for at alle kan se dem. Dette har vist sig at være en stor success.

Pas på dig selv! Det er nemt at brænde sig på et stort projekt.

Hvis man er kreativ og har teknisk kunnen kan man også bevæge sig ud i mere avancerede ting, animationer af alle mulige slags som passer til hvad der sker på scenen. Revy-systemet understøtter en masse seje ting, og der kommer hele tiden flere til så hvis man har mod på lidt programmering er det bare at give sig i kast med effekterne. Men pas på, lad være med at bide over mere end hvad du kan sluge. Alle disse avancerede detaljer er lækkerier der kan oppe oplevelsen, men de kan aldrig stå alene og hvis du ser det som en for stor opgave skal du ikke være bange for at spørge om hjælp eller sige fra.

Men når det er sagt: Man kan kun skabe noget nyt ved at prøve nye ting af.

1.3 Hvad er hvad?

Her følger en række gængse termer der kan være praktiske at bruge, og andre småting der er værd at huske.

Scene: Til hver revy sættes en stor hjemmebygget træscene op i bunden af Store UP1.

Bandscene: En lille forhøjet scene til venstre for scenen, hvor bandet sidder og spiller.

Bagtæppet: Et tæppe til at aflukke til den bagerste meter af scenen, så man kan stille sig klar uset.

Højre: Højre side af scenen set fra TeXnikken og publikum. Den side hvor folk oftest kommer ind fra. (Skuespillere kan ikke finde ud af højre og venstre)

Venstre: Venstre side af scenen set fra TeXnikken og publikum. Den side hvor bandet sidder (Skuespillere kan ikke finde ud af højre og venstre)

Trekanten: Området til højre på scenen bag tæpperne. Hvor skuespillerne kan stå klar og let komme ind fra siden.

TeXnikken: Lyd, Lys og AV, samt diverse hjælpere. Kortsagt dem der sidder foran scenen og arbjeder under revyen

Backstage: Området bag scenen. Bruges også om dem der arbjeder der. De andre "skjulte" roller som dem der laver rekvisiter og kostumer.

Rekvisitten: Dem der laver rekvisiterne/stedet de laver dem (backstage)

Rekvisitkælderen: Revyernes fælles opbevaringslokale, backstage under bandscenen. (Pas på skimmelsvamp)

Dyrestalden: En del af DIKU, gangen der ligger lige når man kommer ud af Store UP1 fra scenen.

Omklædningen: Backstage bag de store metaldøre og indtil trædørene står skuespillernes ting under revyen, gør det svært at komme igennem!

Hyggeområdet: Området bag skuespillernes omklædning, for foden af den sydlige trappe. Her findes ofte snacks og drikkevarer under revyen (loot det i pauserne)

Harlem: Et "hemmeligt" lokale i DIKUs kælder som revyerne råder over. Her øver bandet indtil de rykker ind i auditoriet. Her står også det meste af den texnik som revyerne råder over samt kasser med kabler. Sørg for at spørge før du låner størrere ting, husk at ryde op efter dig, husk at tape kabler sammen inden de dumpes. Spørg din revyboss om adgang hertil

Indre Harlem: Inde i Harlem findes indre Harlem der bruges til at redigere videoer samt de optagne film fra forestillingen. Her står det udstyr vi passer ekstra godt på. Kun de særligt betroede får adgang hertil og man skal spise en skovsnegl og kysse en datalog for at få lov!

Adgang: Husk at snakke med din boss om at få aktiveret dit studiekort så du kan komme ind på DIKU og i Harlem. I TeXnikken kommer og går man ofte på skæve tidspunkter så det er praktisk med adgang.

Instruktør: Dette er meget forskelligt fra revy til revy, men typisk en kreativ/kuntnerisk/skuespilsmæsig ansvarlig for de forskellige numre

Boss: Ansvarlig for hele revyen (bestemmer)

Universitetsparken 1: Stedet der i mange år har huset DIKU (i hvertfald de studerende). Skal forlades i år ....

Store UP1: Det store auditorium på Universitetsparken 1, også kendt som store knirke. Det er her alle revyerne på natfak afholder deres revyer.

Caféen?!: En af fredagsbarene på Natfak. Mange efterfester holdes her, og ofte sender man publikum til Caféen?! om fredagen efter første forestilling/generalprøven.

Overtex: Et stort hvidt område over scenen under loftet hvor AV projekterer det meste af sine ting

Højtex: Et fancy begræb for alle de projektioner der er over Overtex på selve loftet

Lavtex: Et fancy begræb for alle de projektioner der er under Overtex, altså ned på selve scenen (antagelivis på et lærrede der opstilles undervejs)

Fisk: Et fancy ord for en kort film der ligger imellem to numre. Bruges ofte som et sjovt indslag nogen kom på samt til at give lidt ekstra tid til at nå et svært kostume/sceneskift.

Datamat: Fysisk implementation af Turings abstrakte maskine, hvad man på Engelsk ville kalde en "computer".

LaTeX: Et "sprog" til at skrive og opsætte tekst, tænk HTML men til at lave nydelige artikler, formler og pdf'er. Også brugt til at lave overteksters.

TeX: Forgængeren som LaTeXer bygget på, de to termer bruges oftest synonymt.

Emacs: Gammelt tekstredigeringsprogram (Tænk notepad, men mere avanceret end Photoshop). Kan udvides helt utroligt og danner basis for DIKUrevyens AV-system.

Script: Et kort program, en sekvens af instruktioner for hvad der skal ske (automatisk) når det bliver kørt.

Lisp Et gammelt programmeringssprog. Der findes mange varianter men det er kernen i Emacs og udgør dermed måden man laver scripts på i AV-systemet

1.4 Forudsætninger (Skal ændres)

Note: TODO: Dårligt afsnit. Find ud af hvor det passer, og omforumler det til det her skal/vil du lære i bogen

Hvad skal man kunne for at bruge DIKUrevyens AV-system?

- LaTeX
- Linux
- Emacs
- Emacs Lisp
- Python

LaTeX
Bruges til at skrive selve overteksterne. Der bruges Beamer til at lave et langt slideshow, `overtex.sty` definerer en række makroer der gør det let at lave en lang

præsentation. De fleste på natfak burde kunne LaTeX, hvis ikke er den mængde der bruges til at lave normale overtekster ret lille og burde kunne mestres ved bare at kigge i nogle af de gamle filer.

Emacs
Emacs bliver brugt som kontrolcenter til det hele. Det er derfor væsentligt at have en hvis forståelse for hvordan Emacs virker. Emacs er et voldsomt konfigurerbart program til at arbejde med tekst. Jeg er selv stor Emacsbruger og har opbygget et helt unikt system. På sigt er det håbet at lave en standard konfiguration som folk der ikke er emacsbrugere kan udnytte. En sådan konfiguration ville kunne skjule at det overhovedet er emacs der kører bag det hele.

Emacs Lisp
Emacs Lisp er det primære scripting sprog der bliver brugt. Det bruges til at automatisere ting i Emacs, derudover kan der kaldes externe kommandoer og kommunikere med andre maskiner. Selve revysystemet bruger en stor del Emacs Lisp så hvis noget stopper med at virke er det godt at kende. Til almindeligt AV brug kan man dog nøjes med en stærkt begrænset del som denne guide nok skal introducere.

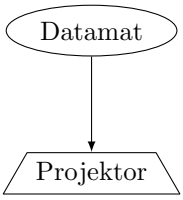
Linux
DIKUrevyens AV-system er bygget og kører på linux styresystemet. Det betydder at man for at bruge systemet effektivt er nød til at have en hvis forståelse for at bruge en linuxmaskine igennem en terminal. Et håb er en gang at have en form for standard opsætning af datamater hvor alt nødvendigt er installeret, som AVmand skal man så blot sætte udstyrret op og lave indholdet.

Python
Bliver brugt til de forskellige programmer og værktøjer der udgør resten af AV-systemet. Det burde kun være nødvendigt at kunne kode python for at udvikle/-vedligeholde systemet.

2 Tanken bag

Den grundliggende filosofi bag systemet er at AV skal kunne komplementere eller understøtte en revy uden at komme i vejen. Der skal dermed kun vises det som publikum skal se og høre og intet andet. Det kan være med til at bryde indlevelsen og virke utroligt amatøragtigt lige så snart publikum ser en cursor, en film der maksimeres eller rammerne på et vindue. Det er sådanne fejltagelser der hiver folk tilbage til virkeligheden i auditoriet fremfor den fiktion man skaber. Man skal ikke tænke over at det er AV, men at det er en mega fed revy, folk bør således kun lægge mærke til AV når der sker noget ekstraordinært, og forhåbentligt ikke på grund af noget der ikke skulle ske.

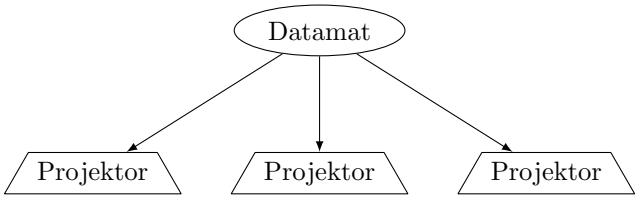
Det simpleste AV-system man kan lave (og som er det de fleste bare ville gøre uden at tænke videre) er at have en datamat tilkoblet en projektor. Datamaten kan derfra køre et presentationsprogram, f.eks. powerpoint.



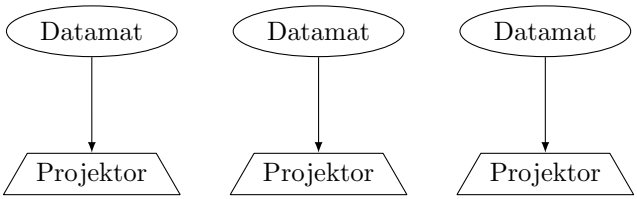
Figur 1: Simpel AV opsætning.

Dette er dog ret primitivt. Det er utroligt nemt at komme til at lave fejl, så som lige at få skubbet cursoren hen på et andet desktop eller at filmen spiller på den forkerte skærm. Ting der er sket gang på gang til mange revyer, og som hurtigt får publikum til at råbe *"TeXniken sejler!"*.

Derudover har man et problem hvis man skal bruge mere end 1 projektor.



Figur 2: En datamat, flere projektorer.

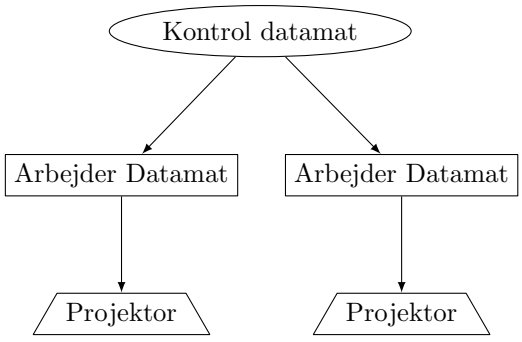


Figur 3: Flere datamater, flere projektorer.

Man kan enten koble flere projektorer på en datamat, hvilket kræver en datamat der er i stand til dette, hvilket ikke særligt mange er. Og det kan give en hovedepine hvis

man samtidigt prøver at holde det "skjult". Alternativt kan man have flere datamater koblet til hver sin projektor. Nu skal man så bare navigere rundt mellem en helt masse maskiner eller have en AV-mand pr. maskine hvilket heller ikke er særligt praktisk. Desuden har vi stadig problemet med at man let kommer til at dumme sig på samme måde som i den første opsætning.

Det vi i virkeligheden ønsker os er en abstraktion mellem det at *styrre* AV og det at *vise* AV. Hvis tingene kører adskilt og har klart definerede ansvarsområder kommer man ikke lige så let til at lave fejl. Vi ønsker derfor at have en central maskine der styrer det hele, denne kommunikerer med andre maskiner der sørger for at vise ting via projektorer.



Figur 4: Central styring, flere arbejdere med hver deres projektor.

Det er dette princip DIKUrevyens AV-system benytter sig af.

Note: Lidt baggrundshistorie for de nørdede.
Det første hjemmelavede AV-system blev lavet af Troels Henriksen. Han lavede et programmeringssprog *Sindre* i Haskell til at lave grafiske grænseflader. I det sprog lavede han et `dmenu` lignende program kaldet *sinmenu* som han brugte som interface til systemet. Selve overteksterne bestod af en pdf som han lokalt oversatte tilbage til rå tekst som blev fodret ind i et script der brugte hans grænseflade. Dette script kommunikerede med en server koblet til en projektor over en `ssh`-forbindelse. `Xpdf` kan køres som en server der kan modtage kommandoer fra en kommandolinje, på denne måde kunne man derfor styre overtekster over ssh.

Desværre skalerede løsningen ikke særligt godt. Da der begyndte at komme mange AV-effekter kunne man ikke både køre overtekster og andet AV alene.

Dette blev (i hvertfald forsøgt) rettet op på da jeg (Søren Pilgård) efterfølgende begyndte at skrive et nyt AV-system til DIKUrevyen 2012. At skrive et AV-system er dog ikke en nem tjans og det endte med at foregå i en løbende process med forbedringer til hver revy.

Den føte forbedrede udgave udskiftede interfacet man som AV-mand arbejdede i. Tanken var at istedet for at arbejde med pdf'en og hive teksten ud af dette, kunne man istedet bruge et værktøj der arbejde direkte på LaTeX koden og så fjernstyrede pdf-læseren ud fra dette. Emacs blev valgt som base for systemet. Det er et system jeg allerede var bekendt med og som jeg i forvejen benyttede til at arbejde med LaTeX. Med Emacs kunne man derfor benytte samme program

til at skrive overteksterne og til at vise/styre dem. Den store force ved Emacs er at det er ekstremt programmerbart, man kunne derfor nemt bygge av-systemet ovenpå. Ligeledes var det muligt for Emacs at kigge i LaTeX koden mens den blev fremvist for at finde indlejret kode som ikke blev vist, men som Emacs kunne evaluere for at gøre andre ting samtidigt. Dette kunne f.eks. bruges til at afspille lyde under en sang eller vise en video.

Hurtigt viste det sig at være et stærkt værktøj at kunne bruge Emacs til at styre hele revyen. Der viste sig dog en ulempe i at bruge forskellige programmer til visningen af hver ting. Man kunne f.eks. ikke vise to forskellige billeder ovenpå hinanden. Ligeledes var der problemer med at en del programmer tog tid at starte op, og at pdf-læseren var lang tid om at reagere på at Emacs bad den om at skifte slide. Et halvt til et helt sekunds forsinkelse lyder måske ikke af meget, men er besværligt når man skal time en linjes overtekst til en hurtig sang.

Derfor opstod idéen om at bygge et dedikeret program til at vise alt hvad Emacs (masteren) kommanderede den til. En første udgave blev udviklet der tog udgangspunkt i java-frameworket `https://processing.org/` som så kunne programmeres til at vise forskellige ting. Til sin sketch kunne man så lave et lille javaprogram som man kunne fjernstyre fra Emacs. Så blev der programmeret en pdf-viser som Emacs fjernstyrede til at vise overtekster. Ligeledes blev der programmeret en billede viser og en lyd afspiller osv. Det viste sig dog ikke at være den bedste løsning. I praksis er det meget få af de numre der har behov for AV der har specielle ønsker, som oftest er det bare en kombination af at vise billeder, lyde, videoer og pdf'er. At alting kørte som forskellige programmerede dele gjorde så at man ikke nemt kunne opbygge AV'en ud fra allerede definerede dele men skulle opbygge alting fra bunden lige så snart man ville bare en lille smule.

Derfor endte det med at en helt anden backend til arbejderne blev programmeret. Idéen var at man let skulle kunne kombinere allerede programmerede dele til sit behov. Det blev besluttet at denne backend skulle programmeres i elisp, eller i det mindste et sprog der mindede meget om dette. På den måde var der således ikke den store forskel på at scripte ting der kørte i Emacs på masteren , og ting der kørte i backenden på arbejderne.

Det er dette system der fungerer som DIKUrevyens AV-system nu.

Note: Skal følgende merges med ovenstående?

DIKUrevyens AV-system består af en central grænseflade der kan kommunikere med flere forskellige maskiner der kan vise AV materiale.

Selve grænsefladen er udviklet som en række udvidelser til Emacs. Grunden til dette er at da jeg startede indså jeg at systemet skulle kunne følgende ting:

- Kommunikere med en server der viser indholdet.
- Det skulle kunne vise ting overskueligt i en grafisk grænseflade.
- Der skulle være mulighed for at indlejre scripts så man kan køre ting automatisk på bestemte steder i forestillingen.

- Det ville være praktisk hvis man kunne rette fejl i texkoden mens man viste pdf'en da man ellers risikere at glemme dem.

Det gik hurtigt op for mig at det ville være fjollet at udvikle noget fra bunden da Emacs i forvejen kunne meget af dette. Desuden er jeg Emacsmand og så hurtigt hvordan en integration ville være nice. Den grundliggende formel for at vise overtekster er at man åbner et LaTeX dokument der bruger beamer pakken til at lave overteksterne. Så starter man `ubertex`minor modet, dette sørger for at pdfen bliver lagt op på serveren. Herefter skjules de fleste LaTeXkommandoer og et overlay lægges der viser hvad der bliver vist. Man kan så trykke "næste" hvilket rykker overlayet ned og synkroniserer serveren til at vise det tilsvarende slide i pdfen. Man kan også trykke vilkårlige steder i tex filen og rykke direkter hertil i overteksterne. Desuden kan der indsættes kode der eksekveres når man når til det pågældende slide.

Derudover findes minormodet `uberscript` der lader en afvikle scripts. Det kunne f.eks. være en sketch med en række lydeffekter eller et kald til en video. Et script kunne også være aktoversigten hvorfra man ved at trykke "næste" kommer ind i det næste nummer, og når dette er færdigt kommer man så tilbage og er klar til næste.

Hvis alt går som det skal, skal man som AVmand kun trykke på en knap (næste) for at afvikle en revy. Dette må være essensen af et godt AV system, når man kun skal tage sig af timingen på skuespillerne/sangerne samt disses fejltagelser.

3 - Guidelines til en AV mand

TODO: Her kommer der til at være en række mere generele råd om hvordan man laver god AV

Det vigtigste man skal huske for at være en god AV-mand, er at have det sjovt! Hvis du ikke har det sjovt, bliver det ikke en sjov revy. Så lad være med at stresse for meget, ignorerer når de andre sejler og glemmer at give dig hvad du skal bruge og hyg dig. Hvis du først lader dig blive presset af det hele mister du overblikket og kreativiteten, så hellere sige "Det går nok" og tage det som det kommer, det skal nok blive en revy.

I dette kapitel findes en række råd og guidelines. Hvordan man laver AV er en smagssag og afhænger meget af hvad man laver og sammenhængen. AV er således en kunst, og selvom det der står her kan lyde som regler, ved enhver kunstner hvornår man skal bryde dem. Nogle gange bryder man reglerne for at få tingene til at gå op i en højere enhed, andre gange for at lave et helt nyt regelsæt. Selvom du måske er anarkist og vil gå din egne veje som AV-mand vil jeg dog anbefale dig at læse og forstå hvad der står her.

Note: Husk der er mange måder at lave AV på, det her er nogle af de erfaringer der har virket godt

Note: Sæt altid højere standarder til dig selv end andre

3.1 Om at lave gode overtekster

I afsnit 5.3 kan du læse mere om hvordan man via systemet let kan konvertere fra et .tex manuskript til overtekster.

Note: Det nytter ikke at have to ens slides, del op i 2x2 eller 1x4

3.2 Om at vise overtekster

Note: Noget om timingen

Husk at hvis man først begynder at vise linjen når sangeren begynder at synge den, så er man i sagens natur allerede bagud.

Detter

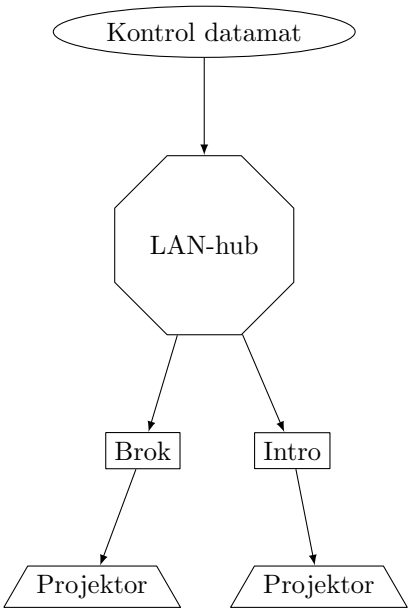
Husk at tage det roligt, når man er i tvivl om ting

Kan være rigtigt at fortsætte når de bare "knævrer"

4 - Opsætning af tekniken

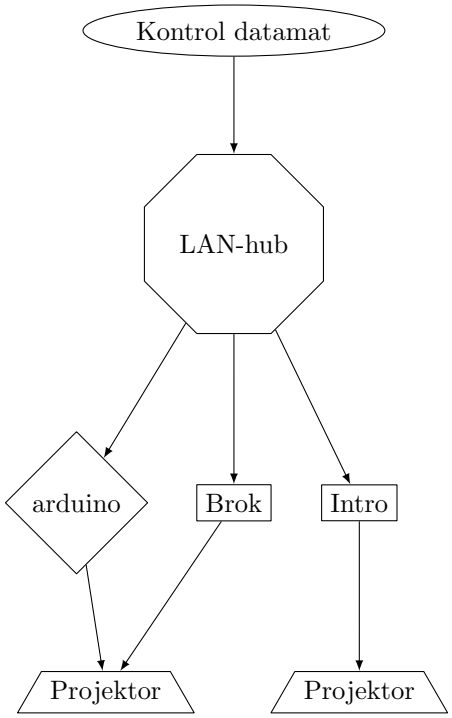
Note: Dette er en meget gammel gennemgang

Til en standard opsætning ala DIKUrevyens skal du bruge:



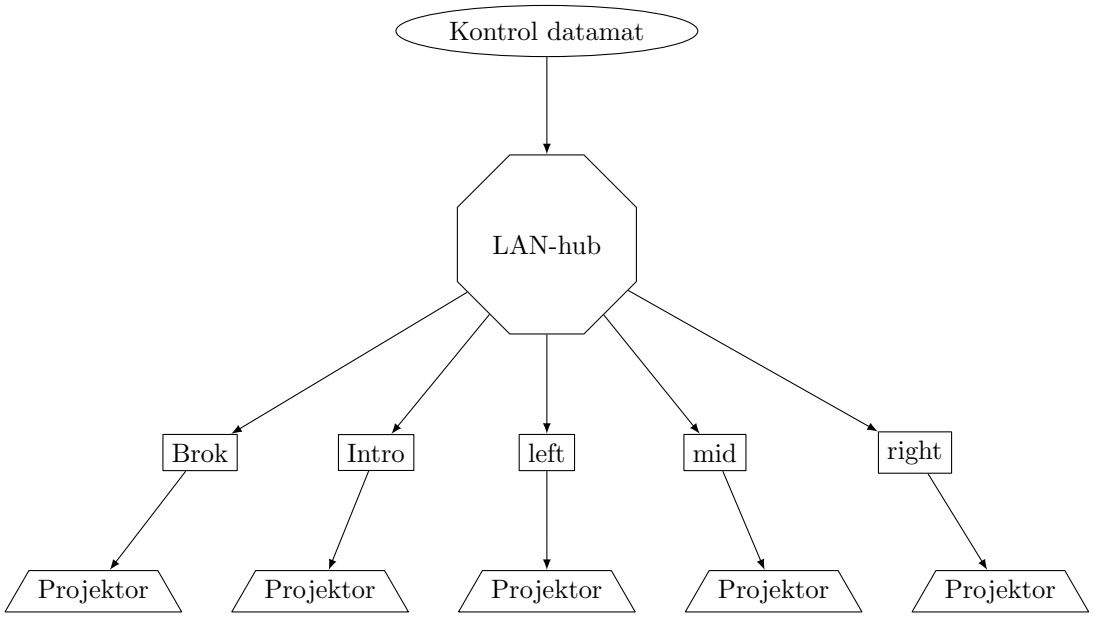
Figur 5: Standard opsætning

Hvis det ønskes (Når det er færdigt) kan man køre med fjernstyrede projektorklapper.



Figur 6: Standard opsætning, med arduino til projektorklap

Systemet kan udvides, her ses f.eks. en opsætning med højtex (uden projektorklapper) som brugt til DIKU Jubilæumsrevy.



Figur 7: Højtex

4.1 Kontrol

Dette er din primære indgang til systemet. Jeg anbefaler at man bruger en Foldedatamat, gerne ens egen bærbare. En bærbar har den fordel at skærmen kan indstilles til både at sidde ned og stå op. Derudover har den tastatur og mus indbygget så det ikke fylder i den ellers rodede texnik. Og så kan man tage den med sig så man kan arbejde videre andre steder. F.eks. er det praktisk at kunne plugge den ud så man kan arbejde videre på sine overtekster til et ellers kedeligt senemøde i hyggehjørnet.

4.2 Projektorer

Til DIKUrevyen bruges der typisk i hvertfald 2 projektorer. Der bruges én der peger op på lærredet over scenen, her vises der overtekster, almindelige av-ting til sketches samt små film.

Kantinen ejer en stor Epson som bruges til introen/av på scenen

DIKUs projektorer bruges til alt andet. DIKU har en række forskellige projektorer, hvor det kan være svært at kende forskel på en del af dem.

En god tradition, som en AV-mand bør holde i hævd, er at rense projektornes filtre når man henter dem i begyndelsen af revyugen. Der er tilsyneladende ikke andre der gør det, så lad det blive en del af rutinen. Så overopheder de ikke lige så nemt. Husk at sætte filteret rigtigt i igen, der er ofte mange tapper og dibedutter der skal passes ind i filteret for at det sidder korrekt.

Til projektorene er der bygget en række projektorkasser af gamle colakasser. Disse gør det en del nemmere at indstille projektorene.
I gamle dage, da jeg var ond. Da blev projektorene stablet på bøger til de stod sirligt, den ærede AV-mand Troels Henriksen brugte mangt en stund på at bande og svovle når disse blev rykket
Nu gør projektorkasserne det en del nemmere da man kan stripse/tape kasserne fast og det hele bliver langt mere stabilt. Desuden kan der komme langt mere luft til.
I gamle dage, da jeg var ond. Da blev projektorne så varme at de overophedede, så vi måtte til HCØ og hente tøris til køling. Det gav også kolde drikkevare (til tider frosne).

Se afsnittene om Ubertex, Uberscript og Emacs for at finde ud af hvordan softwaren bruges.

Der er et forlænger VGA kabel. En ide er at sætte det i krok og føre det over til projektoren, så kan man nemt skifte hvilken maskine der er koblet til projektoren.

Note: TODO: Indsæt diagram fra bagsiden af side 10, Den Store AV-bog printet 27. september 2016

4.2.1 Projektorklapper

En ulempe ved projektore er at deres "sorte" ikke er mangel på lys, men bare *mindre* lys. Det betyder at når alt lyset i StUP1 er slukket og en projektor står og lyser, er der stadig ret meget lys på scenen. Det ser **MEGET** dumt ud og gør at man kan se

hvad der sker på scenen. Det er primært et problem for projektoren til overteksterne og kan accepteres til f.eks. højtex (da loftet ikke er lige så reflekterende)

Det er desværre ikke en løsning at slukke/tænde projektorne da det tager for lang tid/er besværligt/ skydder farver op når de tændes.

Derfor bruges en 'projektorklap'. Der har i mange år været brugt en halv papkasse på en stang. Det fungerer, men det kan godt være lidt stressene. Man skal huske at få klappen på når man er færdig med en sang og har travlt med at huske hvad der nu skal ske. Til tider sker det også at man glemmer at tage klappen af, man når typisk at panikke lidt når der ikke kommer noget billede frem. Det kan betyde at man misser de første par overtekster eller starten af en film.

For at løse dette er det planen at der bygges nogle arduinoer der kan kobles på netværket, disse kontrolerer en klap foran projektoren. På denne måde kan projektoren automatisk begynde at skyde billedet op på lærredet Arduino ftw!

4.3 Netværk

Note: Numerer datamaterne i [1; 254] Hvor xxx er tallet til maskinen Vis f.eks. et eksempel

4.3.1 Statisk IP

Hvis dit netværks interface hedder enp0s25

Note: How to know?

```
sudo ip link set enp0s25 up
sudo ip addr add 192.168.0.XXX/24 dev enp0s25
```

alternativt

```
sudo ifconfig eth0 192.168.0.XXX
```

4.3.2 Hosts

Da det kan være svært at huske alle ip-addreserne kan man i stedet navngive maskinerne. Dette gøres lokalt i `/etc/hosts` et eksempel:

Note: Indfør noter, skelne mellem eksempel, og default værdier. Nævn at arbejdere har en standard statisk IP som de bruger automatisk.

```
192.168.0.20    intro
192.168.0.30    brok
192.168.0.40    left
192.168.0.50    mid
192.168.0.60    right
```

controller: 192.168.0.10 overtex istedet for intro?

Nu kan man ssh'e ind ved blot at skrive:

back: 192.168.0.31 Back(bach): Back mirror of Brok

```
ssh brok
```

Det virker også med scp ol.

4.3.3 Forbindelse uden login

For at logge ind på systemerne over netværket bruges ssh. Da det bliver jævnt irriterende hele tiden at skulle taste løsener kan man lægge sin offentlige ssh nøgle ind på de forskellige maskiner

```
cat .ssh/authorized_keys | ssh revy@192.168.0.30 "cat >> ~/.ssh/authorized_keys"
```

det kan være du først skal oprette mappen `.ssh`.

alternativt kan du bruge

```
ssh-copy-id revy@192.168.0.30
```

Udskift `revy` og ip'en med den relevante bruger og ip. Husk at du skal have en ssh nøgle først dannes med:

```
ssh-keygen
```

4.3.4 ssh mount

I stedet for at scp'e alt muligt crap frem og tibage kan man benytte sig af et ssh mount, som er et filsystem over ssh. Jeg anbefaler at man har filerne liggende lokalt og fra hver maskine der skal kommunikeres med køres `sshfs` som får filerne frem.

Installer sshfs F.eks.:

```
sudo pacman -S sshfs
```

fuse `sshfs` benytter fuse

```
sudo modprobe fuse
```

Hvis dette ikke virker kan det være fordi kernen er blevvet opdateret, men ikke genstartet. I så fald, kan du prøve at genstarte maskinen.

4.3.5 Internet gateway

Hvis du er intereseret i at få internet på det lokale netværk kan man lave en gateway. Dette består i at en af datamaterne på det lokale netværk også er forbundet til et netværk med internet, f.eks. eduroam.

Der er en guide på `sigkill.dk/writings/guides/gateway.html`

På gateway datamaten sættes først den statiske ip (allerede gjordt i forrige trin) Her efter køres gateway.sh som root (hentes på siden)

På clienterne der vil udnytte gatewayen sørger man først for at der er en statisk ip.

herefter skrives

```
ip route add default via 192.168.0.XXX
```

eller alternativt

```
route add default gw 192.168.0.XXX
```

Hvor "192.168.0.XXX" er ip'en til gatewaymaskinen

herefter sættes en dns server, f.eks. google

```
echo nameserver 8.8.8.8 > /etc/resolv.conf
```

Hvis dette ikke virker kan man evt. kigge på `https://wiki.archlinux.org/index.php/Internet_Share` for en ip baseret løsning

Note: Baser hele afsnittet på dette (ip)

uddyb her

Note: internet0 = wlp3s0
net0 =enp0s25

4.4 Arbejdere

Arbejdere er de maskiner/servere der er sluttet til projektorne. Det er deres rolle at vise AV-indholdet det kunne være overtekster, film, lydeffekter, slides, billeder osv.

På nuværende tidspunkt bruges `mplayer` til at vise film, `feh` til at vise billeder og `xpdf` til at vise pdf'er. Det smarte ved `xpdf` er at det kan startes som en server som kan modtage kommandoer via en terminal.

Ideen er at man fra sin kontroldatamat kan ssh'e ind på arbejderne og afvikle den kommando man skal bruge. Emacs med ubertex/uberscript tilbyder basalt set et interface til at gøre dette nemt.

Det er planen at programmet `Zeigen` udvikles. Dette skal erstatte `feh` og `xpdf`. I stedet for at skulle ssh'e ind lytter `Zeigen` til en port og udfører kommandoer på givne klokkeslet. Så kan man sige "om 1/2 sekund skal du afspille denne video" på 3 forskellige maskiner, hvorved videoen vises simultant via `mplayer`.

Et eksempel på en arbejderdatamat er `brok` der bruges til overtekster og ting der skal vises på lærredet. Derudover findes `left`, `mid` og `right` der bruges til at vise *højtex*.

4.5 Hvordan virker en arbejder

Og hvordan bruger man dem. "plug and play" auto netværknetctl + revynet default statiske ip'er Forhåbentligt skal man som almindelig AV-mand rode så lidt som muligt, kun sætte maskinerne fysisk op og tænde, så ordnes resten via masteren iform

af Emacs. Indimellem kan det dog være nødvendigt med maintanence (Det ordner Søren).

4.6 Brok (Deprecated)

Her er en guide om at få brok til at virke:

Sæt ham til Projektor, tastatur, netværk og strøm.

Hvis der skal spilles lyd fra brok bør der være jord på da der ellers kan komme en brummen.

Brok har på skrivende tidspunkt ubunut 11.04 natty. Det betyder også at der er gnome, unity og ting og ækle sager på. Disse har en tendens til at gøre livet surt, da de overskriver xorg konfigurationer og gør mærkelige ting man ikke helt kan gennemskue.

Dette er IKKE optimalt! Support til 11.04 udløb oktober 2012. Der bør installeres et ordentligt styresystem. UDEN gui (installeres seperat).

1: Tænd brok
Brok logger automatisk ind med brugeren ***revy*** med løsnet ***hamster*** Herefter starter unity der har en 'kør' dialog ting.

2: Start en terminal
Skriv `terminal` og tryk enter. Der vil nu komme en terminal op i hjørnet.

3: Kom på netværket

```
sudo ifconfig eth0 192.168.0.30
```

Giv ham en passende ip du kan huske.

4: Opret SSH forbindelse Der kører allerede en ssh daemon. forbind til ***revy*** med løsnet ***hamster***. Du kan evt. uploade din offentlige ssh-nøgle så du slipper for at logge ind.

```
ssh revy@192.168.0.30
```

5: Stop gdm
Gnome Desktop Manager skal lukkes.

```
sudo service gdm stop
```

Dette lukker hele den grafiske grænseflade, inklusiv X.

6: Start screen

```
screen
```

screen køres så vi kan starte X i baggrunden. Når programmet startes vises der en menutekst, bare tryk enter. `screen` kører nu.

7: startx
Start X manuelt.

```
startx
```

Dette starter X op hvorefter .xinitrc eksekveres.

Som det er lige nu startes `xmonad` som windowmanager. Det betyder at skærmen pr. default er sort når x er startet

8: detach fra screen
Tryk `Ctrl-a d`. Dette lader alt der kører i screen fortsætte upåvirket mens du kommer tilbage til det forrige miljø. for at reattache skriv da `screen -r`.

9: test xmonad
Test om xmonad virker. På broks tastatur trykkes `Alt-Shift-Enter`. En terminal burde åbne sig. Luk igen med `Alt-Shift-c`.

Læs evt. afsnitet om xmonad.

Da terminalen er en default `xterm` med hvid baggrund er dette et snedigt trick til at se hele området projektoren kan lyse op. Jeg bruger dette ofte når jeg tweaker projektoren.

10: prøv ting af Med en ssh forbindelse åben, lad os prøve om vi kan få noget til at virke. Start med at vælge det 'display' der skal vises grafiske ting på

```
export DISPLAY=:0
```

Dette skal køres for hver gang du ssh'er ind.

start nu xpdf med en af de gamle overtekst filer der ligger et eller andet sted.

Se sektionen om kommandoer for nærmere detaljer

Hvis man gerne vil lave flere ting simultant på brok kan man sagtens åbne flere lokale terminaler og ssh'e ind parallelt.

4.6.1 Ting der køres på brok

For at hacke uden om alt muligt gøjl køres et par scripts gennem `.xinitrc`

swarp
`swarp` er et program der flytter musse-markøren (cursoren) til et givent koordinat. Swarp findes på `tools.suckless.org/swarp`. `swarp` findes desuden i arch's repository, det kan være den også findes til debian baserede styresystemer, potientielt i en suckless pakke.

`swarp` køres på brok med argumenterne 20000 20000, hvilket flytter markøren ad helvede til.

istedet for `swarp` kan man bruge `unclutter` med kommandoen `unclutter -idle 0 -root &` Som i fjern cursoren efter 0 sekunders delay efter bevægelse inklusiv når cursoren er over rod baggrunden (altså ikke kun over vinduer).

fixsleep `fixsleep.sh` er et script der forsøger at forhindre X i at slukke skærm outputet. X bruger et system der hedder DPMS (Display Power Management Signaling) til automatisk at slukke skærm outputet efter en periode uden tastaturaktivitet.

fixsleep benytter følgende to kommandoer

```
xset dpms 0 30000 40000
xset s 30000
```

den første linje dækker over `xset dpms [standby [suspend [off]]]` den anden linje dækker over `xset s [timeout [cycle]]`

keepon

```
xset dpms force on
```

Denne kommando forcer dpms fra, (det svarer til at at trykke på en tast)

`keepon.sh` er et script der ligger i baggrunden og kører denne kommando whert 30 sekund.

fixsleep og keepon forsøger begge at holde dpms stangen ved at lave aktivitet. Man kan måske istedet bruge

```
xset -dpms; xset s off
```

Til at slå dpms fra. De to systemer kan dog ikke blandes.

Man kan se om dpms er slået til ved at kalde `xset -q`

4.7 Xmonad

Windos/super eller alt som modifier knap.

5 - Master

Se også 7.3

5.1 ubersicht

5.2 ubertex

5.3 Konvertering fra manuskript til overtekster

%Emacs (Deprecated, merges med Master)

Note: Denne section blev fundet i en gammel udprintet udgave af den-store-av-bog, men kunne ikke findes i git historikken. Den er noget udtateret i forhold til hvad vi gerne vil og antager lidt at folk har styr på ting.

Her er en kort introduktion til den del af revy-systemet der foregår i Emacs. Emacser ikke bare en teksteditor men et helt miljø til at arbejde med tekst og evaluere lisp kode (specifikt en dialekt kaldet Emacs lisp eller bare elisp)

Som AV-mand er en stor del af ens arbejde i bund og grund at arbejde med den tekst der skal vises og afvikle den kode der viser det, derfor er Emacs et så centralt element i systemet. I appendix XXX er en introduktion til at bruge en forsimplet udgave af Emacs. Hvis man synes at Emacs er lidt besværligt at bruge kan man istedet kigge på "Simple Emacs" som er en konfiguration af Emacs designet til revyen og som minder om et mere traditionelt program (det kan dog stadig anbefales at sætte sig ind i hvordan Emacs virker).

Note: Nej der er ej

Revyens AV-system er designet som en række filer indeholdende forskelig funktionalitet. Som udgangspunkt er det nok at tilføje `ubertex/emacs` mappen til sin `load-path` og sw indlæse "revy"

```
(add-to-list 'load-path "path/to/ubertex/emacs")
(require 'revy)
```

Som udgangspunkt defineres blot to funktioner `revy-create` og `recy-load` Først når disse bruges indlæses resten af funktionaliteten. På denne måde slipper man for at have mere end højst nødvendigt indlæst hvis man benytter Emacs til andre formål.

Da Emacs ikke har "namespaces" eller en anden form for modulsystem er det med vilje designet med så *få* hjælpe funktioner som muligt. Tanken er at enhver funktion virker selvstændigt så man som bruger af systemet kan se på enhver funktion kaldet `revy-` og bruge den i sine egne scripts. Dette betyder dog at enkelte funktioner kan blive lidt lange og indviklede, det er en pris der må betales for at det bliver så simpelt at programmere op i mod som muligt.

Implementerings noter?

5.4 revy.el

Indeholder grundfunktionaliteten. De to funktioner `revy-create` og `revy-load` henholdsvist opretter og indlæser en revy. Herefter indlæses revyen samt funktionerne til at arbejde med denne.

Det er ikke nok bare at åbne en `.revy` fil manuelt.

5.5 uberrevy.el

Binder revy modesne sammen. Overordnet er en række variabler defineret her. Når en revy indlæses, loades uberrevy som sørger for at loade alt andet.

5.6 ubercom.el

Indeholder funktionalitet til at kommunikere med leiter (som står for kommunikationen med arbejderne).

Derudover er en række hjælpefunktioner defineret til at gøre kommunikationen med arbejderne lettere. Disse kan både bruges interaktivt eller i koden.

5.7 ubersicht.el

Ubersicht er et system til at holde styr på scripts. I bund og grund er det et system bygget til at evaluere et emcas script skridt for skridt. Dette sammenholdt med en lang række revyspeciffike funktioner giver en stor fleksibilitet.

Hvert skridt består af en *instruktion* ("instruction" i koden)

Man kan bruge ubersicht til at lave en fil med aktoversigten, ved hvert nummer benytter man så ubersichts "næste" funktionalitet til at bevæge sig videre til næste nummer, dette kan så være en kommado der åbner f.eks. en sang eller en sketch. Man kan ligeledes benytte ubersicht til at have en sekvens af instruktioner der afvikles under en sketch. Det kunne være at der undervejs i sketchen skal afspilles nogle lyde, billedr eller film, disse vil være skrevet ind i filen som en række instruktioner. Når man i sketchen når til at det er tid til at afvikle en instruktion trykker man blot "næste. Til sidst vil man typisk have en "afslut" instruktion der returnerer tilbage til det ubersict scrpit der åbnede det nuværende, det kunne f.eks. være aktoversigten.

Systemet er lavet fleksibelt for at gøre det bredt anvendeligt men en typisk brug består i én aktfordeling liggende i en fil kaldet `revyens-navn.revy` eller noget i den stil. Indholdet kunne se således ud:

```
(revy-start)


Akt 1
(revy-open "sketches/velkommen.sketch")
(revy-open "sange/en_sjov_sang.tex")
(revy-open "sketches/sketchen_med_ordspil.sketch")
(revy-open "video/dum_video.sketch")
(revy-open "sange/nu_er_der_pause_sangen.tex")

Akt2
(revy-open "sange/revyen_er_sjov.tex")
(revy-open "sketches/dårlig-sketch.sketch")
(revy-open "sange/sidste_sang.tex")

Ekstranumre
(revy-open "video/en_anden_video.sketch")
(revy-open "sange/sjovt_ekstranummer.tex")
```

Den første instruktion man evaluerer er at starte revyen, herefter kører man igennem de forskellige numre i revyen.

En instruktion er et kald til en elisp funktion og består **altid** af en startparentes på en ny linje. Dermed bliver teksterne "Akt 1", "Akt 2" og "Ekstranumre" ignoreret. Der er dermed rig mulighed for at sætte noter ind. Man kan skrive elisp kommentarer der begynder med et `;`, man kan skrive elisp strenge `"Noget tekst"` eller blot skrive klartekst (Pas dog på, hvis filen evalueres som normal elisp kode vil klartekst indskrevet direkte i filen blive evalueret som kode, hvilket kan give en fejl). eller makro Som i *-config.el

En markør, makerer hvilken instruktion der bliver udført på nuværende tidspunkt. Markøren farver hele instruktionen så det er tydeligt. Husk at markøren (kaldet "cursor" i koden) ikke er det samme som tekstmarkøren hvor tekst bliver indsat når der tastes (denne kaldes i Emacs for "point").

Overordnet er der 3 funktioner relateret til at bruge ubersicht

`revy-ubersicht-mode`
Er et Emacs major mode, baseret på det indbyggede emacs-lisp-mode. Dette bruges primært for sine genvejstaster samt at holde styr på den nuværende sketch. Når man kalder funktionen startes major modet.

`revy-ubersicht-next`
Rykker markøren frem og udfører den næste instruktion. (Rykker også point).

`revy-ubersicht-enter`
Eksekverer instruktionen som point er i ellerden efterfølgende.

Derudover har vi adgang til en lang rzkke funktioner der kan bruges som instruktioner. I virkeligheden kan enhver elisp funktion bruges. En række funktioner til brug for at styre og afvikle revyen er defineret og findes i `uberinstructions.el`

5.7.1 Instruktioner der er værd at kende

`revy-start`
Starter en revy.

`revy-open`
Åbner og begynder eksekveringen af et nyt script eller overtekster.

`revy-nop`
Gør ingenting og ignorerer alle argumenter. Imodsætning til en kommentar eksekveres denne instruktion stadig, den udfører bare ingenting. Dette kan bruges hvis f.eks. er en sketch uden AV-indhold, man kan på denne måde stadig følge med i hvor man er i revyen ved blot at trykke næste.

(alternativt kan man bruge `ignore` som gør næsten det samme.)

`revy-end-sketch`
Afslutter den nuværende sketch og returnerer til den der åbnede den. Hvis man fra sin aktoversigt.revy har kaldt `revy-open` på en sketch, vil et kald til `revy-end-sketch` i sketchen afslutte denne og hoppe tilbage til aktoversigten så man kan fortsætte til næste nummer.

`revy-shell`

Er defineret i `ubercom.el`, og kan afvikle en shell kommando på enten den lokale maskine eller på en arbejder.

TODO: denne funktion skal refaktoreres

5.8 ubertex.el

Ubertex er på mange måder tilsvarende uberscript, men hvor man i et uberscript benytter sig af elisp funktioner til at udføre en sekvens af instruktioner er ubertex beregnet til at vise slidesne til en pdf skrevet i LaTeX med beamer klassen.

Pointen er at man skriver en latex fil med beamerkalssen og overtex pakken. Man kan der definere hvert slide, samt pauserne id disse. Når nummeret skal vises kører en pdf fremviser på arbejderne som styres via ubertex i Emacs. Systemet minder om uberscript, en markørvires hvilken del af latexen der vises som slide i pdf'en, og man kan så bladre ned igennem.

```
\documentclass[14pt]{beamer}
\usepackage[danish]{babel}
\usepackage[utf8]{inputenc}
\usepackage{overtex}

%% Melody: Den der disney sang, https://www.youtube.com/watch?v=dQw4w9WgXcQ

\begin{document}
\obeylines

\begin{overtex}
  % forspil
\end{overtex}

%% S1
\begin{overtex}
  Det her er første linje\pause
  den er lang så vi deler den
\end{overtex}

\begin{overtex}
  % blank
\end{overtex}

\begin{overtex}
  Mere sang\pause{} med en pause\pause
  det er vældigt smart
\end{overtex}
\begin{overtex}
  Flere linjer\pause
  i denne sang
\end{overtex}
```

```
\begin{overtex}
  Tralala\pause
  Tralala\pause
  Tralala\pause{} lalala
\end{overtex}

\begin{overtex}
  % slut
\end{overtex}
\begin{overtex}
  \elisp{(revy-end-sketch)}
\end{overtex}
\end{document}
```

Når man afvikler en sang, bliver latexken skjult af vejen så man kan fokusere på indholdet. Husk at naår man laver en ændring i latexen livestreames den ikke til pdf-fremviseren, først skal latexkoden oversættes til en pdf, herefter skal denne overføres til arbejderne der så skal have at vide at nu skal den nye pdf vises istedet.

5.9 manus.el

Indeholder brugbar funktionalitet til at konvertere et `.tex` manuskript til brugbare overtekster.

`revy-manus-mode` Starter `revy-manus-mode`. Primært er formålet at stille en række genvejstaster til rådighed.

5.9.1 Konvertering fra manuskript til overtekster

```
(add-to-list 'load-path "~/code/ubertex/emacs")
(require 'revy)
```

6 - Worker

Note: Generelt om hvordan en worker virker

Se også 7.4

7 - Lisp

Lisp er en famillie af programmeringssprog, en af de ældste udgaver er Emacs Lisp, eller bare elisp som bruges til at udvide og opbygge programmet Emacs. Elisp er et ældre sprog, fra 1985, derfor kan mange ting godt virke lidt specielle og "sære", men overordnet fungerer det godt til formålet. Hvis du kender til lisp i forvejen vil det hjælpe en hel del, men elisp minder som sprog mere om common lisp end om scheme, racket eller clojure.

Til av-systemet bruges elisp til at programmere masteren. En variant kaldet wlisp kører på workerne. wlisp er designet til at minde meget om elisp, og i praksis behøver du typisk ikke at tænke dem som to forskellige sprog. Der er dog nogle små forskelle hist og her, disse vil blive nævnt i dokumentationen herunder.

Ok, men hvad er et programmeringsprog? Et programmeringssprog er en måde at sammensætte en sekvens af instruktioner som en "computer" kan forstå. Du kan se det som en måde at lave en opskrift, men istedet for at forklare hvordan du skal bage en æblekage, forklarer vi hvordan en arbejder skal tegne et billede hvor vi ønsker. Der findes mange forskellige programmeringssprog der kan se meget forskellige ud, men som alle bygger på nogle fælles underliggende træk.

Udover denne guide til lisp har Emacs en kæmpe mængde dokumentation af alle mulige systemer til rådighed. I Emacs kan du trykke `C-h i` for at få en liste over alle disse. Der er en guide til at bruge Emacs, men derudover er der en "Emacs Lisp Intro" til at forklare elisp for begyndere, desuden er der en stor section om elisp der dokumenterer alle de elisp dele Emacs kommer med som standard.

I Emacs menu, som enten vises i toppen af vinduet som de fleste andre programmer, alternativt hvis denne menubar er slået fra (som mange gør) kan man få den frem midlertidigt ved at holde `Ctrl` nede og så højreklikke. I menuen "Help" kan du i undermenuen "More manuals" også finde elisp introduktionen samt elisp referencerne.

7.1 Delene i Lisp / De forskellige lisp dele

7.1.1 Værdier

I elisp og wlisp fremover bare nævnt som lisp har vi et koncept om "værdier". En værdi er noget konkret som f.eks et heltal, et komma tal, en tekst streng, en liste eller et symbol, derudover findes kode der kan arbejde med disse, det kunne f.eks. være funktionen `+` der tager et par tal og returnere et andet tal. Alt kode returner en værdi, nogle gange bliver værdi ignoreret, f.eks. fordi den er ligegyldig. Nogen kange kan kode gøre mere end bare returnere en værdi, så siger vi at det har en "sideeffekt" fordi det har en effekt udover bare at returnere en værdi. En sideeffekt kunne være at vise et billede, afspille en lyd, åbne en fil eller noget helt andet. En af de ting der gør lisp lidt specielt er at det er ret nemt at lave kode der arbejder med andet kode, fordi vi faktisk kan se kode som en værdi som "evalueres" for at gøre noget. Vi kan styre denne evaluering og hvornår den sker hvilket er et væsentligt koncept i lisp, og som bruges af av-systemet.

Der findes flere slags værdier, disse "slags" kaldes en type. I lisp har heltal typen `integer` og kunne f.eks. være 13, -17, 0, 1074234. Bemærk at for effektivitet er

heltal begrænset til en endelig mængde det betyder at vi ikke kan repræsentere tal større end 2147483647, eller mindre end -2147483647. Denne begrænsning er typisk ikke et problem i de programmer vi laver her.

Et kommatal af typen `float` er en slags reelt tal. 9.17, 1.0, -14.30421, 0.1 er alle `float`s. Vær opmærksom på at der er en begrænset præcision, tallet vil derfor altid være afrundet til en værdi der kan repræsenteres internt i computeren. Du kan derfor ikke gemme π med alle decimaler, men må nøjes med en mindre størrelse. Denne afrunding gør at `float`s tiltider kan føre til resultater der ikke helt matcher hvad du forventede, det er derfor typisk en dårlig idé at bruge `float` hvis du vil have et eksakt resultat at sammenligne med. Men til meget almindeligt brug er de en fin type.

En `string` er en tekststreng. En sekvens af bogstaver, tal og tegn der bruges som en tekst. Det kunne f.eks. være et filnavn, en tekst der ønskes skrevet på skærmen, eller noget andet. En streng kunne se således ud `¨Hello, World!¨` bemærk de to dobbeltplinger/quotes som bruges til at angive at indholdet imellem er teksttrengen. Hvis man gerne vil have en sådan til at forekomme inde i strengen kan man "escape" den med et "backslash". `¨abc\¨def¨`

Et `symbol` er et ord og minder måske lidt om en `string`, men et symbol er kun et enkelt ord, og indeholder altså ingen mellemrum. Man laver typisk et symbol med `'` operatoren `'fisk`. Symboler er en af grundstenene i lisp, deres formål er typisk i at når de evalueres returnerer de ikke sig selv, men en anden værdi. En variabel er således et symbol der når det bliver evalueret resulterer i en anden værdi.

En liste skrives som `(1 2 3)` en start parantes efterfulgt af indholdet separeret af mellemrum. Lister bliver intern omsat til hægtede lister i form af såkaldte "cons" celler. En conscelle er en værdi i computerens hukommelse der indeholder to værdier. Et hovede og en Hale (tænk på en haletudse). Consceller kan bruges til alle mulige formål, men deres klassiske brug er til lister. I en liste er konventionen at det første element i listen findes i hovedet på den første conscelle, halen peger på den anden conscelle. Det andet element findes i hovedet på den anden conscelle og den tredje conscelle bliver peget på i halen på den anden. Således er hele listen opbygget af consceller der peger på værdier og på resten af listen. I den sidste celle er der jo ikke flere celler tilbage, derfor indeholder halen symbolet `nil` for at angive at der ikke er flere elementer tilbage. En alternativ måde at skrive `nil` på er som `()` altßw de tomme parantesser, eller den tomme liste. `nil` og `()` bliver behandlet som akkurat det samme, det er bare to måder at skrive det samme.

Note: TODO: lav et diagram

Lister er ligesom variabler lidt specielle, de evaluerer ikke til sig selv men til et funktionskald. Koden `(+ 1 2)` bliver derfor evalueret som et kald til funktionen `+` med argumenterne `1` og `2`. Hvis man gerne vil have en liste kan man istedet bruge `list` funktionen7.4 eller man kan bruge ' operatoren som siger at den efterfølgende operand ikke skal evalueres `'(1 2 3)`

Når vi i lisp taler om et udtryk taler vi om en sådan værdi, eller mere præcist noget som når det bliver evalueret resulterer i en værdi. Således er en liste der indeholder flere lister bare ét udtryk (men opbygget af flere andre udtryk). Ligeledes er et funktionskald der som argument tager resultatet af flere andre funktionskald, også ét udtryk med flere udtryk som argument.

7.1.2 Sandhed

De fleste programmeringssprog har et koncept om sandheden af en værdi. Mange sprog har en "boolean" type til at konkretisere værdierne sandt og falsk, sådan er det dog *ikke* i lisp. I lisp har vi det lidt specielle symbol `nil` som repræsenterer den falske værdi. I lisp er `nil` falsk som den eneste værdi, alle andre værdier betragtes som at være sande. Husk på at `nil` også er den tomme liste, det vil sige at lister med indhold er sande, mens den tomme liste er falsk.

Om noget er sandt eller ej bruges i betingelser, som `if`, `when`, `unless` og `while` der alle gør forskellige ting alt efter om et af deres argumenter evaluere til en sand eller falsk værdi.

7.1.3 Variabler

En variabel minder måske lidt om dem du kender fra matematik, men ikke helt. Variabler bruges til at huske ting, enten midlertidigt mens vi arbejder med noget, eller for at huske noget på længere sigt. En variabel er et navn for en værdi, så når man bruger variablen bruger man faktisk den værdi der er bundet til den. En variabel har et navn, et symbol, som er bundet til en konkret værdi på et givent tidspunkt i programeksekveringen. Når lisp evaluerer et symbol slår den op i en tabel over alle variabler og finder ud af hvad variablen er bundet til, så returneres denne værdi. Lad os sige at variablen `alder` er bundet til værdien `42`, så kan vi skrive `alder` i koden og dette vil returnere værdien `42` når det bliver evalueret. Hvis du vil vide hvor gammel man bliver om et år kan man så f.eks. skrive `(+ alder 1)` når lisp evaluere dette vil det først evaluere symbolet `alder` til værdien `42`, så evalueredet heltalet `1`, dette evaluere til sig selv, altså `1`, så evalueres funktionen `+` med argumenterne 42 og 1, dette kald returnere så resultatet 43 som den endelige værdi.

Så hvis vi skriver `x` evalueres dette af lisp til den værdi `x` er bundet til. Hvis vi bare gerne vil have symbolet `x`, skal vi istedet skrive `'x`

For at tildele en værdi til en variabel kan man bruge `setq` som i `(setq alder 44)` dette sætter variablen med navnet `alder` til værdien 44. `setq` står for "set quoted" dette er en makro der ikke evaluerer navnet men tager det uevalueret for så at sætte den tildelte værdi i tabellen over alle variabler. I dette tilfælde evalueres `alder` altså ikke til den tidligere værdi, og fordi `setq` er en makro behøver vi ikke at skrive `'alder`. Der findes også en anden funktion `set` som ikke er en makro, og den *skal* kaldes med et symbol som i `(set 'alder 45)` denne funktion bruges dog ganske sjældent i normal kode.

7.1.4 Funktioner

En funktion er noget kode der kan eksekveres (udføres) når funktionen bliver kaldt. En funktion minder på en måde om funktioner som du måske kan huske fra matematik $f(x) = 3x^2+7x-4$, her er det beskrevet hvordan funktionen f afhænger af parametret x og hvordan resultatet bliver udregnet, vi kan nu udregne at $f(8)$ giver værdien 244. Hvis vi gerne ville formulere denne udregning som en funktion i lisp kunne vi skrive:

```
(defun f (x) (+ (* 3 x x) (* 7 x) -4))
```

Og vi kan som før udregne at `(f 8)` giver resultatet 244. Bemærk hvordan funktionsnavnet står som det første inde i parantesen. En funktion er således en måde at beskrive hvordan noget udregnes, muligvis afhængigt af nogle parametre. En funktion er dog lidt mere generel, den kan nemlig også have "sideeffekter". I virkeligheden ville det måske være mere korrekt at kalde dem procedurer end funktioner, det tager vi dog ikke så tungt her og holder os til den mere gængse terminologi. Det betyder dog at en funktion kan mere end blot at beskrive hvordan man udregner et resultat, de kan faktisk bruges til at beskrive vilkårlige sekvenser af hvordan ting skal gøres. Det kunne f.eks. være at tegne et landskab ud fra nogle billeder af træer, en himmel, et hus og en mark.

Når vi kalder en funktion i lisp kan vi give nogle argumenter med, f.eks. gav vi argumentet 8 til funktionen `f`. Disse evalueres og gemmes så i parametret som en variabel, herefter udføres kroppen af funktionen med parametrene. I ovenstående eksempel er `x` derfor en variabel som er bundet til 8 når vi kalder funktionen med 8 som argument.

Så kan lisp gå i gang med at evaluere kroppen. I dette tilfælde består kroppen af et kald til funktionen `+`, der gives her 3 argumenter til denne funktion, først evalueres første argument, det er et kald til funktionen `*`. Vi giver 3 argumenter til `*`, først tallet 3 som evaluere til sig selv, herefter `x`, `x` er en variabel og evaluere til det den er bundet til, i det her tilfælde 8, herefter evalueres `x` igen, dette udfra at $x \times x = x^2$. Nu udregner lisp kroppen af multiplikationen der giver resultatet 192, dette er altså det første argument til `+`, nu udregnes andet argument `(* 7 x)` som ligeledes er et kald til `*`, 7 evaluere til 7, `x` evaluerer til 8, og produktet er 56, som det andet argument til `+`, det tredje argument er tallet -4 som evaluere til sig selv. Nu har lisp altså udregnet de tre argumenter til `+` funktionen som altså svarer til `(+ 192 56 -4)` dette kan lisp så udregne og det giver 244. Nu kan lisp så se at der ikke er mere at lave i funktionen `f`, derfor returneres den seneste værdi som resultatet af hele funktionen, `f` returnere således 244 tilbage til der hvor den blev kaldt.

`+` er en lidt speciel funktion, den kan tage vilkårligt mange argumenter som den summerer, ligeledes kan `*` bruges til at udregne produktet af alle sine argumenter

Det kan ses som meget arbejde, men det hele gøres automatisk af lisp når vi eksekverer koden, et program er således opbygget af en masse kald til forskellige funktioner der typisk kalder andre funktioner indtil alle værdierne er udregnet. Det kan virke lidt uoverskueligt i starten, men denne måde at bygge funktioner op på er ret central i lisp (og de fleste andre programmeringssprog), det tilader os at opbygge et lag af abstraktioner, vi kan nu snakke om at udregne f frem for at snakke om multiplikation og addition, eller det at tegne et landskab frem for at tegne enkelte dele.

Note: Hvad er forskellen på et argument og et parameter? parametret er navnet på variablen/variablerne som funktionen tager dem, det er altså de navne værdierne bliver bundet til. Når vi kalder en funktion giver vi den nogle værdier, disse kaldes argumenter. Argumenter bruges altså når vi kalder funktioner, parametre bruges i funktionens definition.

I praksis er det dog ikke specielt vigtigt hvad du kalder dem, mange bruger de to udtryk uden at tænke over hvad det præcis er, og ingen vil tænke synderligt over det om du siger argument eller parameter.

Måden vi erklærer en funktion er med `defun`, dette er en makro som erklærer en

funktion, hvilke parametre den har og hvad den gør når den bliver kaldt. Det første argument til `defun` er funktionens navn, dette skal vi bruge hver gang vi fremover vil kalde funktionen. Det andet argument er en liste af parametrene funktionen har, som udgangspunkt skal en funktion kaldes med et argument til hvert parameter funktionen har, man skal derfor give præcis det samme antal argumenter som funktionen har parametre. Efter listen af parametre kommer der potentielt en såkaldt "docstring", dette er en kort dokumentation af hvad funktionen gør. Docstringen er valgfri, til sin korte funktion der skal tegne et landskab til revyen behøver man den næppe, men langt de fleste funktioner i Emacs har f.eks. en sådan dokumentation.

Herefter kommer selve kroppen. Dette er en vilkårlig lang sekvens af udtryk, når funktionen evalueres, evalueres disse udtryk et ad gangen. resultatet af evalueringen af det sidste udtryk bliver returneret som resultatet af hele funktionskaldet.

Udover de obligatoriske "normale" parametre kan man også have valgfrie argumenter. Til dette bruger man det lidt magiske ord `&optional` i sin sekvens af parametre, efter dette er de efterfølgende parametre valgfrie. Det betyder at når man kalder koden skal man angive alle de obligatoriske parametre, og man kan, men behøver ikke at angive de valgfrie argumenter. det kunne f.eks. se således ud:

```
(defun foo (a b &optional c d)
  (list a b c d))
```

Funktionen med det intetsigende navn *foo* skal kaldes med to argumenter, men man kan kalde den med ialt 2, 3 eller 4 argumenter. De argumenter man kalder funktionen med bliver gemt i parametrene i rækkefølge. De parametre der ikke bliver givet som argument bliver tildelt værdien `nil`, altså den intetsigende falske værdi. Dette er så måden at se på om der ikke blev givet et argument.

Vi kan f.eks. kalde `(foo 1 2 3 4)` og `list` kaldet i `foo` vil lave lave en liste ud af argumenterne og resultatet vil da være `(1 2 3 4)`. Hvis vi kalder `(foo 1 2 3)` er resultatet `(1 2 3 nil)`, hvis vi kalder `(foo 1 2)` er resultatet `(1 2 nil nil)`.

Når vi ikke giver et argument med er parametret således sat til `nil`, vi kan bruge dette til at lave en anderledes opførsel for en funktion:

```
(defun bar (&optional a)
  (if a
      "Jeg fik en værdi"
    "Jeg fik ikke en værdi"))
```

Her har vi en betingelse, (mere om dem i 7.2.1) hvis `a` *ikke* er `nil` så returneres strengen `¨Jeg fik en værdi¨` ellers, hvis `a` er `nil`, returneres `¨Jeg fik ikke en værdi¨`. Så hvis vi kalder `(bar)` returnerer den `¨Jeg fik ikke en værdi¨` og hvis vi kalder `(bar 10)` returnerer den `¨Jeg fik en værdi¨`,

Der er dog en ting vi skal være opmærksomme på, vi kan ikke skelne om et argument ikke blev givet, eller om `nil` værdien blev givet som argument. Hvis vi kalder `(bar nil)` vil resultatet derfor være `¨Jeg fik ikke en værdi¨` Dette kan være lidt overraskende da man jo faktisk gav en værdi med. Man kan vælge at se det som at hvis man kalder en funktion med `nil` som argument til et valgfrit parameter,

så siger man at man explicit bare vil have at den skal gøre det som den gør som standard uden et argument. Det kan være brugbart for en funktion der tager flere forskellige valgfrie parametre og har forskellig opførsel alt efter værdien af disse. Så kan man f.eks. give `nil` som værdi for at få den til at behandle nogle argumenter som om vi ikke gav dem, samtidigt med at vi giver nogle af de senere explicit. F.eks. kunne vi kalde `(foo 1 2 nil 4)` som ville returnere `(1 2 nil 4)`, nu er dette et ret simpelt tilfælde, men det er praktisk for nogle af de mere komplicerede funktioner i Emacs.

En anden type valgfrie parametre er "resten" parametre, disse kan bruges til at sige at en funktion kan tage vilkårligt mange argumenter. De bliver så givet som et rest parameter i en liste indeholdende dem alle. Man bruger også her det lidt magiske ord `&rest` foran et enkelt parameter navn som alle de resterende argumenter bliver gemt i.

Vi kunne dermed oprette vores egen udgave af `list` funktionen vi så tidligere:

```
(defun my-list (&rest l)
  l)
```

Denne funktion har ét rest parameter kaldet `l`, som alle argumenterne til `my-list` bliver lagt i. Funktionen gør ikke andet en at returnere denne liste. Vi kan så kalde `(my-list 1 2 3)` og vi får `(1 2 3)` som resultat, eller vi kan kalde `(my-list 1 2 "a"3 "c"4)` og vi får `(1 2 "a"3 "c"4)` tilbage som resultat. Hvis vi kalder `(my-list)` får vi den tomme liste tilbage `nil` (som er det samme som `()`).

Vi kan også kombinere dette med obligatoriske argumenter

```
(defun my-list-atleast-2 (a b &rest l)
  (cons a (cons b l)))
```

Denne funktion sætter de to første argumenter sammen i et par consceller der peger på hinanden og videre på resten af argumenterne i `l`, således konstrueres der én lang liste. Resultatet er altså lig det fra `my-list`, forskellen er at denne funktion fejler hvis ikke den bliver kaldt med 2 eller flere argumenter.

Vi kan også have valgfrie parametre:

```
(defun my-optional (a b &optional c &rest l)
  (cons a (cons b (cons c l))))
```

Denne lidt mærkelige funktion *skal* have mindst 2 argumenter, ellers fejler den, man *kan* vælge at give den et tredje, ellers indsættes et `nil` i listen, og man kan give den vilkårligt flere der så indsættes i enden. `(my-optional 1 2 3 4 5)` giver så `(1 2 3 4 5)` `(my-optional 1 2)` giver `(1 2 nil)` og hvis vi kalder `my-optional` med færrere end 2 argumenter resulterer det i en fejl.

Det giver **ikke** mening at have flere parametre efter `&rest` nøgleordet, men man kan navngive det som man vil.

Note: Vær opmærksom på at først skal du angive **ALLE** obligatoriske parametre, så skal du angive **ALLE** de valgfrie, og herefter kan du angive "resten", du kan altså ikke blande rundt i rækkefølgen

En sidste ting omkring funktioner, er at man skal være opmærksom på at de først kan bruges *efter* de er defineret. Og at det altid er den seneste definerede udgave der kaldes. Hvis man omdefinerer en funktion behøver man altså ikke at gøre noget ved de funktioner der allerede er defineret og som kalder funktionen, de vil automatisk kalde den nyest definerede udgave.

7.1.5 Makroer

En makro minder på mange måder om en funktion, en central forskel er dog

7.1.6 Fejl

elisp undtagelser, wlisp error ikke definerede variabler giver fejl.

7.1.7 Lokale variabler

Der findes forskellige slags variabler. Dette har noget at gøre med hvor og hvor længe de er synlige/eksisterer. En normal variabel hvis vi bare skriver `(setq foo 10)` vil være en global variabel. Denne er synlig i resten af programmet af alt kode. Vi kan således skrive `foo` for at få den tildelte værdi. Hvis vi nu efterfølgende skriver `(setq foo 11)` vil vi opdatere værdien for foo til 11, vi kan således ikke længere se den gamle værdi 10, men istedet den nye værdi 11.

Til tider kan det være praktisk med en eller flere midlertidige variabler, til brug mens man udregner noget, men som ikke har noget formål at gemme. Så kan vi bruge `let` eller `let*` . Disse introducere nogle *nye* variabler der kun eksisterer inde i kroppen af let udtrykket. Hvis vi bruger `;=>` som en syntaks i vores kode eksempler til at betyde "evaluere til" så gælder der da at

```
(setq foo 10)
foo ;=> 10
(let ((foo 100))
  foo ;=> 100
  (setq foo 101)
  foo ;=> 101
)
foo ;=> 10
```

Så `foo` sættes til at være værdien 10. I let udtrykket defineres så den nye variabel der også hedder foo, denne har værdien 100. Den originale variabel findes stadig, men når vi nu skriver `foo` henviser vi altså til den nye udgave og ikke den gamle. Dette gælder også hvis vi ændrer i værdien, dette gøres for den nuværende variabel, og ikke den gamle. Efter let udtrykket henviser `foo` altså igen (eller stadig) på den gamle variabel 10.

Denne måde at lave en midlertidig binding er utrolig praktisk, på den måde undgår vi at overskrive anden data som vi faktisk gerne ville beholde Det er især praktisk i forbindelse med funktioner. I en funktion vil vi gerne undgå at et kald har en unødig indvirkning på anden kode. Ved at bruge et let udtryk ungår vi dette. Her er et par eksempler:

```
(defun foo ()
   (setq a 1)
   (bar)
   a)
(defun bar ()
  (setq a 2)
  a)
(foo) ;=> 2
```

Her henviser `a` til den samme globale variabel. Derfor returneres værdien 2.

```
(defun foo ()
   (let ((a 1))
     (bar)
    a))
(defun bar ()
  (setq a 2)
  a)
(foo) ;=> 1
```

Foo erklærer en ny variabel `a`, men på grund af "dynamisk scope" er det denne variabel som `bar` ser og opdatere. Det endelige resultat er derfor det samme, men efter kaldet til `foo` vil den globale værdi af `a`, hvis den findes være den samme som tidligere.

```
(defun foo ()
   (let ((a 1))
     (bar)
    a))
(defun bar ()
  (let ((a 2))
    a))
(foo) ;=> 1
```

Dette er den korrekte måde at undgå at variabler i andre funktionskald ændres. `foo` erklærer en ny variabel `a`, det samme gør `bar`, i `bar` vil alle ændringerne af `a` inde i let udtrykket derfor ikke være synlige i `foo`.

En anden ting der er omkring variabler er at parametrene til en funktion effektivt er midlertidige variabler defineret og synlige i funktionskroppen og bundet til værdien af det tilsvarende argument.

```
(defun foo (a)
  (setq a 1)
  (bar a)
```

```
  a)
(defun bar (a)
  (setq a 2)
  a)
(foo 0) ;=> 1
```

Noget om component lokale variabler Noget om ikke definerede variabler

7.1.8 ...

Note: Vectorer er ikke ens. I Emacs lisp er de konstante. I Worker Lisp er de dynamiske arrays. ak indsætte foran og bagi i O(1) tid (ammortiseret)

Lister er en slags "meta" type over consceller. En konvention! En liste er en Conscelle hvor `car` indeholder et element i listen, og `cdr` indeholder en liste (resten). Alternativt er en liste `nil` som angiver den tomme liste eller slutningen på en liste. Der er således ikke en konkret type i sproget der hedder *list*, men man bruger det som om der var.

Quote opfører sig anderledes (Skal jeg lade være med at deep copy'e?)

7.2 Hvordan programmerer man?

7.2.1 Betingelser

7.2.2 Løkker

7.2.3 Hvordan arbejder man med lister

7.2.4 Om at finde/rette fejl

7.3 Lisp på master

Dette er en liste af kommandoer der kan bruges på masteren. En del af disse kan også bruges interaktivt

7.3.1 Lokale kommandoer

Kommandoer der gør ting lokalt på masteren.

`(revy-start)`
Starter en revy, den første kommando i en .revy fil. Forbinder til alle arbejderne og starter programmet. Hvis den fejler er det typisk fordi en server ikke har forbindelse til netværket (måske er den ikke tændt), fordi masteren ikke har forbindelse til netværket (har du sat stikket i / fået en statisk IP?) eller fordi programmet er installeret i en anden mappe end den angivne.

Note: Der er lige nu en fejl i programmet der gør at denne kommando nogle gange skal køres to gange for at virke

`(revy-quit)`

Lukker programmet på alle arbejderne hårdt og brutalt. Den inverse funktion af revy-start. Bør kun bruges fordi tingene virkeligt er gået i hårdknude, f.eks. hvis en arbejder stopper med at svare når man prøver at eksekvere kode på den.

`(revy-open filename &optional worker)`

Åbner en fil i revysystemet og eksekverer den i dens revysammenhæng. Se de specifikke revy modes5.1, 5.2 for en bedre indføring i de enkelte dele. Når en fil åbnes på denne måde, vises den i en buffer i det korrekte mode og eksekveringen starter fra toppen. Rækkefølgen af filer der åbnes, og deres placering huskes, så et kald til `revy-end-sketch` fra den åbnede fil returnerer til her.

"`filename`" er en tekststreng med navnet på filen. Umiddelbart bruges en relativ sti `(revy-open "sange/lalala.tex")`, og denne sti tager så udgangspunkt i mappen hvor revyen er gemt (på din lokale maskine). Alternativt kan man bruge en absolut sti hvis man f.eks. pludselig fik lyst til at åbne en fil fra en gammel revy.

"`worker`" er navnet på en arbejder, denne vil blive brugt som "default worker" istedet for den nuværende. Så kan man f.eks. åbne en tex/pdf på en anden arbejder end den nuværende

```
(revy-open "sange/lalala.tex" 'mid)
```

"`revy-nop &rest..`" Gør ingenting, og ignorerer alle argumenter. Kan f.eks. bruges til at udkommentere en kommando. Kan også bruges i .revy filen istedetfor `revy-open` for at signalere at dette nummer ikke indeholder noget AV. Imodsætning til en kommentar eksekveres denne instruktion stadig, den udfører bare ingenting. Dette kan bruges hvis f.eks. er en sketch uden AV-indhold, man kan på denne måde stadig følge med i hvor man er i revyen ved blot at trykke næste.

```
(revy-open "sketches/den-sjove.sketch")
(revy-nop "sketches/den-kedelige.sketch")
(revy-open "sange/sjov-sang.tex")
```

Den kedelige sketch markeres stadig når man trykker videre fra den sjove sketch, men der sker ikke noget, så ved næste tryk bevæger man sig videre til den sjove sang.

`(revy-end-sketch)`

Lukker et nummer. Kan bruges i alle slags filer, både .tex og .sketch. Returnerer til den fil der har åbnet den nuværende og fortsætter fra hvor cursoren stod sidst. Bruges typisk til at komme tilbage til .revy oversigten fra en sang eller sketch.

Funktionen kalder `revy-abort-all`, og lukker dermed allt kørende kode på arbejderne automatisk.

`(revy-return)`

Virker lidt lige som `revy-end-sketch`, returnerer til det sted der åbnede det nuværende nummer. Men lukker ikke kørende kode. Den kører altså ikke `revy-abort-all`.

`(revy-restart)`
Genstarter det nuværende nummer. Svarer til at lukke og åbne filen.

7.3.2 Generelle kommandoer

`(revy-abort-all)`
Den store røde stop knap! Stopper alt der kører på alle arbejderne. Alle lyde, billeder, pdf'er og kode der kører afbrydes. Er praktisk at have i baghånden til når tingene går galt, og kan bruges interaktivt til dette. Men kan også bruges inde i en sketch til at stoppe alt der køres samtidigt. Vær opmærksom på at lyde stoppes ret abrupt, hvilket ikke altid er hvad der ønskes. Bør ikke bruges som panik knap hvis f.eks. en sanger synger forkert, i så fald stoppes hele pdf'en med at blive vist. Så er man nødt til at genåbne filen, finde tilbage igen, og fortsætte. Til dette formål bør man istedet bruge "blank" funktionerne.

`(revy-abort)`
Lige som `revy-abort-all`, men stopper kun de ting der kører på den nuværende arbejder. Så den kan f.eks. bruges i en sketch til at holde højtex kørende mens man stopper et billede og en lyd på overtex.

`(revy-blank))`
Stopper visningen af grafik på den nuværende arbejder indtil næste input. Alt kode og lyd fortsætter med at køre, men skærmen "cleares" inden den vises. I praksis svarer det til at der tegnes en uendelig stor sort firkant oven på alting. Blankningen forbliver indtil der modtages en form for kode eller signal. I praksis indtil du gøre noget der skulle gøre noget på arbejderen.

Bruges typisk interaktivt til at stoppe visningen af sangtekster når sangeren synger forkert. Så blanker AVmanden, finder frem til hvor sangeren er nået til, og fortsætter så. Når AVmanden fortsætter holder blankningen op og teksten vises som normalt.

`(revy-blank-all)`
Som `revy-blank`, men blanker alle arbejdere. Vær opmærksom på at arbejderne venter enkeltvist på at få et signal om ikke længere at være blanke.

Bruges ikke så ofte da formålet med at blanke typisk kun er at fjerne det der er forkert, sangerens tekst, og ikke alt hvad der også er på de andre arbejdere.

`(revy-unblank-all)`
Sender signal til alle arbejdere om at de ikke længere skal være blanke. Bruges typisk efter et `revy-blank-all`

`(revy-calibrate)`
Viser et kalibrerings billede. Er en simpel måde at sikre sig at alle arbejdere virker, er tændt og deres projektorer står indstillet korrekt.

7.3.3 Audio Visuelle kommandoer

`(revy-image file &optional position)`

Sender et program til den nuværende arbejder der viser et billede. Den indbyggede `image` funktion på arbejderne tegner et billede på en position en enkelt gang i et enkelt frame. For at få vist et billede i længere tid kan man bruge denne funktion der sender en stump kode der definerer en billede viser der kontinuerligt viser billedet. Den bliver ved med at tegne billedet i hvert frame indtil den afbrydes. `revy-image` sætter bare det nuværende kode til at være denne billede viser.

Se 7.4 for hvordan det præcist virker med at afvikle kode i hvert frame.

```
(revy-image "billeder/hund.png") ;; Viser et billede af en hund.
(revy-image "billeder/kat.png")  ;; Viser et billede af en kat istedet.
```

"Position" argumentet er lidt magisk og kan en hel masse

Note: TODO: Forklar noget om dette

`(revy-image-preload &rest files)`

Bruges ikke til noget.

`(revy-pdf-open file)`

Åbner en pdf fil og viser slide 0 (det første). Til sangtekster vil man typisk bruge ubertex5.2 som også sagtens kan bruges til at lave andre slideshows en bare overtekster. Til billeder og diasshow er det ofte at foretrække at bruge enkelte billeder da de nemmere kan justeres enkeltvist med "position" argumentet for hvert enkelt slide. Men ind imellem får man som AV-mand en pdf med billeder der skal vises. Til dette kan man bruge `revy-pdf-open` som definerer en pdf-viser meget lig den førnævnte billedeviser.

Vær opmærksom på at det kan tage lidt tid at åbne en stor pdf.

Note: Vær opmærksom på at revysystemet 0-indeksere slides i pdf'er. en pdf på 3 sider vil da indeholde slide 0, 1 og 2.

`(revy-pdf-goto-slide slide)`

Skifter hvilket slide den *nuværende* pdf-viser viser. "slide" er et 0-indekseret tal der angiver hvilket slide der skal vises.

Note: Hvad sker der hvis tallet er størrere end antallet af slides?

`(revy-pdf-next)`

Skifter slidet den *nuværende* pdf-viser viser til det næste

Bør generelt ikke bruges i en sketch da man ikke længere kan gå til en vilkårlig position og fortsætte. Hvis en skuespiller f.eks. hopper fra slide 5 til slide 8 i sit diasshow, kan man ikke hoppe med hvis man kun er sat op til at bruge `revy-pdf-next`. Hvis man derimod bruger `revy-pdf-goto-slide` kan man rykke direkte ned til slide 8 og vise dette.

Note: Hvad sker der hvis det nuværende slide er det sidste?

`(revy-sound file &optional volume)`

Afspiller en lyd. Lyden bliver afspillet som en sepperat komponent, så den har ikke indvirkning på anden afviklet kode/lyde. Man kan således bruge revy-sound flere gange til at afspille flere lyde på samme tid. Man kan angive en ønsket volume, et heltal mellem 0 (min) og 128 (maks). Hvis ikke lydstyrken angives spilles den på maks.

Note: Vær opmærksom på at den lydmixer der bruges lige nu er ret dårlig og forringer lydkvaliteten en hel del. Til forestillingen er dette dog ikke det store problem på grund af andet lyd/larm i salen.

Hvis man virkelig ønsker sig at en bedre lydkvalitet, f.eks. ved musikken til et dansenummer kan man istedet for `revy-sound` bruge `revy-mplayer` der udover videoer også kan afspille musik. Så mister man dog en del fleksibilitet til at stoppe/fade lyde.

`(revy-stop-sounds)`

Stopper alle lyde der afspilles på den nuværende arbejder. Virker lidt som `revy-abort`, men kun på lyde.

`(revy-fade-sounds &optional duration)`

Virker lidt som `revy-stop-sounds`, men istedet for at stoppe lydene brat, så fades de langsomt ud. Man kan angive en varighed i sekunder, som standard tager det 4 sekunder at fade ud.

Er et godt alternativ til `revy-stop-sounds` i mange situationer da det ofte lyder bedre.

Hvis man kalder den interaktivt spørger den efter en varighed.

`(revy-mplayer file &optional x y w h)`

Afspiller en video. Kører det eksterne program "mplayer" på den nuværende arbejder som afspiller videoen. Kan også afspille lydfiler.

Note: Hvordan indstiller man mplayer?

`(revy-kill-mplayer)`

Lukker alle instanser af "mplayer" på den nuværende arbejder. Ofte kan man bruge revy-abort istedet.

`(revy-text &optional text)`

Viser en tekst midt på skærmen af den nuværende arbejder.

Kan kaldes interaktivt, hvor den spørge om en tekst der så vises. Man kan bruge piletasterne til at finde tidligere tekster.

Note: Snak om Markup sproget

7.3.4 Interaktive kommandoer

`(revy-create)`
Opretter en ny revy. Fører dig igennem en længere "wizard" (opsætnings guide).

Note: Forklaring om hvordan denne virker

`(revy-load)`
Loader en allerede oprettet revy. Spørger dig om du vil indlæse en af de revyer den kender til, typisk den du arbejder på. Ellers kan du vælge "other revy..." hvorefter du kan angive den præcise placering for en revy.

`(revy-mode-enter)`
`(revy-mode-next)`
`(revy-mode-point-forward)`
`(revy-mode-point-backward)`

7.3.5 Andre

`(revy-on-worker worker function &rest args)`
`(revy-create-workers)`
`(revy-send-lisp)`
`(revy-send-command)`
`(revy-shell)`
`(revy-shell-sync)`
`(revy-shell-local)`
`(revy-shell-local-sync)`
`(revy-elisp)`
`(revy-upload-files)`
`(revy-upload-files-sync)`
`(revy-build)`
`(revy-compile-tex)`

7.4 Lisp på worker

`(update)`
Du skal forstå dette her hvis du vil lave noget som helst avanceret... Så er det ærgeligt at dokumentationen er mangelfuld, beklager...

```
(progn)
(quote)
(eval)
(list)
 (cons)
(car)
(hd)
(head)
(cdr)
(tl)
(tail)
(if)
(when)
(unless)
(while)
(and)
(or)
(print)
(set)
(setq)
(let)
(let*)
(eq)
(equal)
(not)
(defun)
(lambda)
(+)
(-)
(*)
(>)
(sin)
(cos)
(randint)
(color)
(clear-color)
(clear)
```

`(fill)`
`(image)`
`(pdf)`
`(text)`
`(calibrate)`
`(sound)`
`(sound-stop)`
`(sound-stop-all)`
`(sound-fade-all)`
`(defcomp)`
`(defcomponent)`
`(create)`
`(destroy)`
`(current-layer)`
`(deflocal)`
`(update)`
`(render)`
`(send)`
`(broadcast)`
`(receive)`

magiske `(component_update_all)`
`(message_dispatch)`
`(resource_cache_size)`
`(sounds_playing)`
`(allocate_useless)`
`(render_test)`
`(pdf_test)`
`(sdl_internals)`
`(resource_usage)`
`(exit_program)`
`(set_window_position)`

8 - Værktøjer

8.1 Schneider

Tysk: Skrædder

Dependencies:

- Python3
- ffmpeg

Schneider er et værktøj til at skære film og billeder op til at kunne blive vist over flere projektorer.

Schneider kan på skrivende stund kun dele medier op i flere snit ved siden af hinanden og ikke over under. Det burde dog være let at rette til.

8.2 Zeitherr

Tysk: Timelord Er en bastardiseret udgave af en NTP tidsserver til at holde de forskellige maskiner synkroniseret.

Note: Externe:

8.3 xpdf

Overtekster køres i xpdf der åbnes manuelt med:

```
xpdf -remote ubertex -fullscreen -mattecolor black -fg black
    -bg black -papercolor black filnavn
```

Da dette er meget langt kan man istedet bruge aliaset

```
p filnavn
```

8.4 mplayer

Til at vise videoer manuelt bruges mplayer:

```
mplayer -nolirc -msglevel all=-1 -msglevel statusline=5
    -vo gl2 -autosync 30 -cache 1048576
    -cache-min 99:100 -xy 500 -geometry 49%:40% filnavn
```

eller aliaset

```
m filnavn
```

8.4.1 Positionering af mplayer

TODO: Noget om positionering af mplayer her

Note: Konfigurer mplayer fra Emacs, væk på sigt

8.5 Gimp

8.6 Inkscape?

8.7 Audacity

9 - FAQ

Ubertex tager ikke højde for pauser i comments, latex gør.

9.1 LaTeX

9.1.1 Hvordan indsætter jeg ¨ (quotes)

Hvis Emacs indsætter `` hver gang du faktisk trykker ¨ så er det fordi Emacs prøver at være smart og indsætte LaTeXs pæne quotes automatisk. Dette er selvfølgelig irriterende hvis du ønsker at indsætte ¨ til brug i koden, f.eks. i indsat elisp. Hvis du blot trykker ¨ en ekstra gang laves det dog om til det korrekte tegn.

9.1.2 File 'overtex.sty' not found.

`overtex.sty` er en fil hvori de overtex speciffike kommandoer er defineret. `revy-manus-prepare` indsætter automatisk `\usepackage{overtex}` i overtexfilerne. Når man kalder `revy-compile-tex` specificeres automatisk hvor denne fil findes. Kaldes pdflatex manuelt ved den ikke hvor denne fil findes. Den simpleste løsning er at oversætte med et kald til `revy-compile-tex`.

Alternativt kan man enten kopiere `overtex.sty` ind i mappen hvor `.tex` filen ligger, sætte shell variablen `TEXINPUTS` til at indeholde mappen hvori `overtex.sty` ligger eller kopiere filen ind i `~/texmf/tex/latex/overtex/overtex.sty` hvor den vil være synlig for oversætteren.

9.2 Build fejler

9.2.1 Jeg har lige oprettet en revy / Jeg vil gerne se min første overtex

`revy-build` forsøger at oversætte samtlige .tex filer. Hvis bare én af tex filerne ikke kan oversættes, så fejler build. Da tex filerne fra manuskriptet kræver en speciel revy pakke fejler de typisk. Det betyder at man er nødt til at konverterre samtlige .tex filer til overtekster (eller til noget der oversætter) før man kan builde.

9.3 - Problemer/løsninger

9.3.1 Der er lag/forsinkelser

Lag er irriterende, det opleves primært som en forsinkelse fra man har trykket på knappen til der sker noget på projektoren. Det er et problem når man skal lave overtekster.

Til DIKUs jubilæumsrevy var der ca. et sekunds forsinkelte fra jeg trykkede til at overteksterne blev vist. Jeg har ikke definitivt fundet fejlen endnu men der bliver arbejdet på det.

Her er først nogle metoder til at lokalisere hvor omtrentligt forsinkelsen opstår. Det er mere eller mindre umuligt at lave konkrete målinger så det er noget man må føle sig frem til (Super naturvidenskabelig metode!).
Følgende tager udgangspunkt i Xpdf, men gøres på samme måde med Zeigen

Prøv først at sætte systemet til at køre alt lokalt. Aka, kør både Emacs og Xpdf lokalt og der kommunikeres via ssh til `localhost`. Hvis forsinkelsen stadig er der, er fejlen enten i Emacs, i Xpdf, eller i den lokale hardware.

Mine erfaringer med Xpdf er dog at det ikke er her fejlen ligger. Prøv nu at starte Xpdf på serveren og ssh ind på denne, giv nu manuelt Xpdf ordre til at skifte slide.

Min erfaring er at det er meget tilfældigt hvad der præcist sker. Jeg oplevede at delayed forsvandt efter jeg gjorde ovenstående, også det mellem Emacs og Xpdf, men kun indtil Xpdf blev lukket.

Det virker også til at Xpdf bliver langsommer afhængigt af længden af overteksterne og ikke nødvendigvis størrelsen af pdfen.

Jeg ved ikke helt hvordan dette skal løses. Det er en af grundene til at vi vil lave vores egen fremviser (Zeigen).

Note: Noget om youtube playbackRate til at ændre hastighed/speed `x = document.getElementsByTagName`

10 - Eksempler

Note: Med screenshots og eksempler, både på brug af program og effekter Se det som hvad jeg ville vise til et AV-føl Videosekvens på skrift/billede

11 Videregivelse af AV

Du skal oplære et føl!

Den idéelle tid for en texnikker er 4år plus.

i praksis 2-3 år, mange desværre kun et enkelt.

Så oplær et føl...

A LaTeX

LaTeXer et markup sprog der bruges til at designe dokumenter. Man bruger HTML som du måske kender til at lave hjemmesider, LaTeX derimod bruges til at lave dokumenter. Det er originalt lavet til at skrive artikler og bøger, især inden for Datalogi, matematik og fysik, da det har rig mulighed for at opsætte formler pænt. Mange på NatFak bruger LaTeX til at skrive artikler, aflevering og alt muligt andet. Den store AV bog som du læser lige nu er f.eks. også skrevet i LaTeX. Ligeledes bruger en del revyer LaTeX til deres manuskript. Udover at lave artikler kan man også lave præsentationer ved hjælp af beamer pakken. Det er sådan overteksterne kan laves. Der er en AV-specifik pakke kaldet overtex.sty der kan bruges og som indeholder makroer Emacs kan forstå og bruge til at fjernstyre backenden.

Hvis du aldrig har arbejdet med LaTeX før kan det godt virke lidt kryptisk og kompliceret. Du er måske vant til at arbejde med word og power-point som er såkaldte 'What You See Is What You Get" redskaber (wysiwyg). Det betyder at det er det sammme program du bruger om du redigerer eller bare kigger i et word dokument. Du ændrer direkte i det som du ser det. LaTeX derimod er et programmeringssprog skrevet i en helt simpel tekst fil, med endelsen .tex det betyder at du ville bruge et program som notepad til at arbejde med koden, og ikke et program som word. Emacs er et godt bud på et værktøj til at arbejde med tex filer. At det er et programmeringssprog betyder at man ikke bare markere et stykke tekst og trykker **fed** men at man skriver noget kode der beskriver at det her skal læses som `\textbf{fed}` det kan tage lidt tid at vende sig til men det er ikke en uoverkomelig opgave. For så at få en visbar fil, f.eks. en pdf, skal man "oversætte" .tex filen. Dette kan man gøre med `pdflatex`. Emacs og især av-systemet har værktøjer til at gøre dette. Det sker f.eks. når man kalder `revy-build` der oversætter alle .tex filer og overfører dem til arbejderne.

Det kan ske at en .tex fil indeholder fejl, det kan f.eks. være at man har skrevet en af LaTeXs kommandoer forkert, glemt et special tegn, brugt noget der ikke findes eller brugt det forkert. Hvis der er en fejl stopper oversætteren med en fejlbesked. LaTeX har notorisk dårlige fejlbeskeder, og de er ikke lette at tyde eller at finde automatisk via software. Hvis du bruger `revy-build` vil der blive lavet en buffer kaldet "*revy-compile-filnavn*" hvor filnavnet er navnet på texfilen der oversættes. Du kan kigge i denne buffer for at finde ud af hvad der gik galt. Det kan også være en idé at kalde `revy-manus-clean` som kan finde en del almindelige fejl.

Her ses et eksempel på en .tex fil:

```
\documentclass[14pt]{beamer}
\usepackage[danish]{babel}
\usepackage[utf8]{inputenc}
\usepackage{overtex}

%% Melody: Den der disney sang, https://www.youtube.com/watch?v=dQw4w9WgXcQ

\begin{document}
\obeylines

\begin{overtex}
  % forspil
```

```
\end{overtex}

%% S1
\begin{overtex}
  Det her er første linje\pause
  den er lang så vi deler den
\end{overtex}

\begin{overtex}
  % blank
\end{overtex}

\begin{overtex}
  Mere sang\pause{} med en pause\pause
  det er vældigt smart
\end{overtex}
\begin{overtex}
  Flere linjer\pause
  i denne sang
\end{overtex}

\begin{overtex}
  Tralala\pause
  Tralala\pause
  Tralala\pause{} lalala
\end{overtex}

\begin{overtex}
  % slut
\end{overtex}
\begin{overtex}
  \elisp{(revy-end-sketch)}
\end{overtex}
\end{document}
```

Preamplen, alt teksten inden `\begin{document}` er fortæller LaTeX hvordan indholdet skal forstås.

% bruges til at lave en kommentar, alt tekst herfra indtil linjens slutning ignoreres af LaTeX.

`\obeylines` betyder at LaTeX overholder linjeskiftne. Så et linjeskift i .tex filen også betyder at der er et linjeskift i pdf'en. Dette er det mest naturlige i overtekster.

A.1 AV-specifik

```
\begin{overtex}

\end{overtex}
```

Definerer et slide, alle linjerne imellem bliver vist sammen.

`\pause` kan bruges til at dele et slide op i to sider i den endelige pdf. På den måde vises linjerne inden "pausen" først og så de næste linjer bagefter. Det er almindeligt at have `\pause` som slutning på de første linjer i et slide. Men man kan faktisk også have pauser inde i linjerne der så bliver splittet op. Så anbefales det dog at skrive `\pause{}` uden de krøllede paranteser bliver mellemrummet frem til det efterfølgende ord spist/ignoreret.

A.2 Generelt brugbare makroer

Man kan lave forskellige skriftstørrelser. Følgende kommandoer ændrer størrelsen inde i et scope,

\tiny	noget tekst
\scriptsize	noget tekst
\footnotesize	noget tekst
\small	noget tekst
\normalsize	noget tekst
\large	noget tekst
\Large	noget tekst
\LARGE	noget tekst
\huge	noget tekst
\Huge	noget tekst

Et eksempel kunne være følgende taget fra satyr-revy 2016:

```
\begin{overtex}
  {\Huge FUCK\pause{} MIT\pause{} LIV!}
\end{overtex}
```

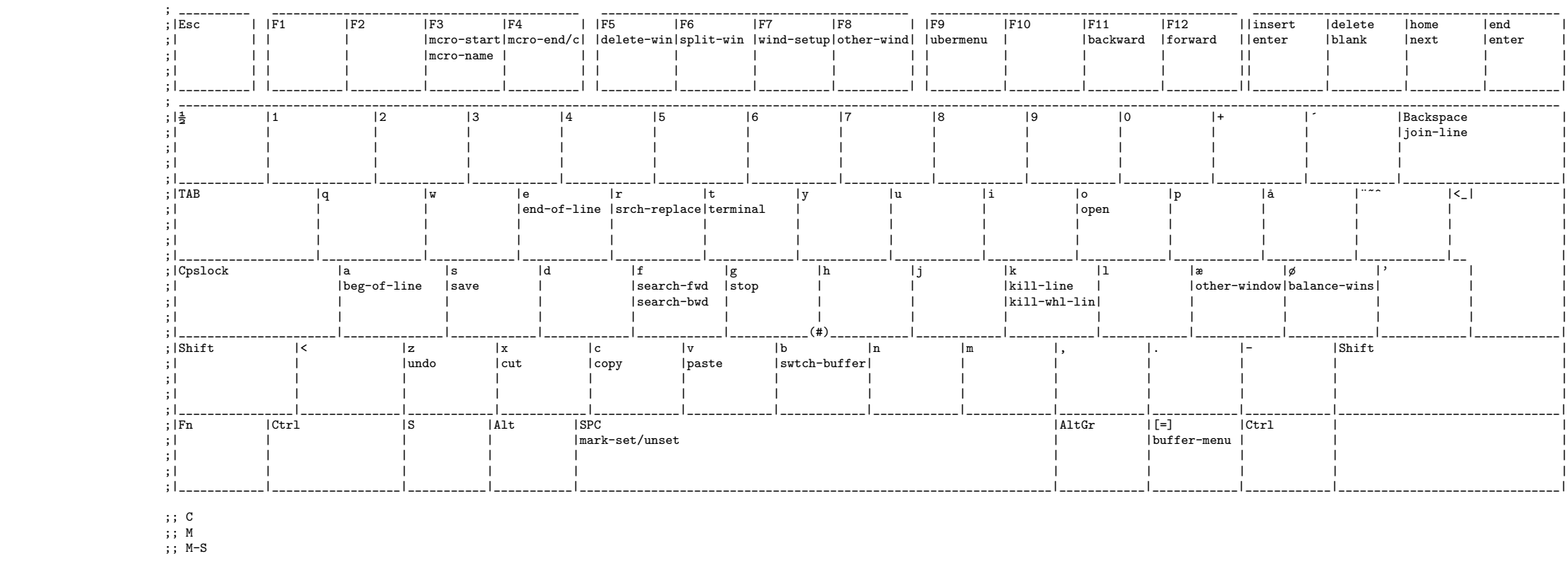
Som i over tre sider skriver teksten med stort.

B Sådan bruges “Simple Emacs”

Emacs kan være svært at bruge. Buffere, genvejstaster, input, hov noget gik galt.

Note: Copy-pastet fra lisp, nævn Emacs tutorial og Emacs manual.

I Emacs menu, som enten vises i toppen af vinduet som de fleste andre programmer, alternativt hvis denne menubar er slået fra (som mange gør) kan man få den frem midlertidigt ved at holde `Ctrl` nede og så højreklikke. I menuen “Help” kan du i undermenuen “More manuals” også finde elisp introduktionen samt elisp referencerne.


Esc
F1
F2
F3 mcro-start mcro-name
F4 mcro-end/c
F5 delete-win
F6 split-win
F7 wind-setup
F8 other-wind
F9 ubermenu
F10
F11 backward
F12 forward
insert enter
delete blank
home next
end enter
1
2
3
4
5
6
7
8
9
0
+
Backspace join-line
TAB
q
w
e end-of-line
r srch-replace
t terminal
y
u
i
o open
p
å
<_|
Cpslock
a beg-of-line
s save
d
f search-fwd search-bwd
g stop
h
j
k kill-line kill-whl-lin
l
æ other-window
ø balance-wins
Shift
<
z undo
x cut
c copy
v paste
b swtch-buffer
n
m
,
.
-
Shift
Fn
Ctrl
S
Alt
SPC mark-set/unset
AltGr
[=] buffer-menu
Ctrl

```
;; C
;; M
;; M-S
```

Her er en gennemgang af mange af de genvejstaster der findes i Emacs, specifikt med den forsimplende konfiguration kaldet Simple Emacs, som findes i `revy-simple`

For at slå den permanent til skal man gå ind i sin `.emacs` fil i sin hjemmemappe og tilføje følgende linjer:

```
(add-to-list 'load-path "~/ubertex/emacs")
(require 'revy-simple)
```

Hvor stien for `load`-path peger på `emacs` mappen i den mappe hvor `ubertex` er installeret.

`<f9>` — *menu*
Åbner menuen

`C-o` — *Open file*
åben en fil i en ny buffer.

`C-s` — *Save file*
gemmer filen.

`C-S-s` — *Save all buffers*
Gemmer alle ikke gemte buffere.

`C-e` — *End of line*
flytter point til slutningen af linjen.

`C-a` — *Beginning of line*
flytter point til starten af linjen.

`C-k` — *Kill rest of line*
sletter alt tekst fra point til slutningen af linjen.

`C-S-k` — *Kill whole line*
sletter hele linjen.

`H-<backspace>` — *Join line*
sletter linjeskiftet før denne linje, så den forige og denne bliver til én lang linje.

`C-f` — *Find*
Find en tekst i bufferen. Når man har trykket tasten, bliver man spurgt om en tekst, Emacs begynder med det samme at søge ned igennem bufferen. Søgningen starter fra der hvor point er lige nu. Hvis man vil søge videre efter næste forkomst kan man trykke `C-f` igen. Hvis man når til slutningen af bufferen vil `C-f` starte forfra fra toppen. Når man er færdig med at søge skal man trykke `<enter>` som stopper søgningen. Man kan trykke `C-S-f` for at *reverse* søgeretningen og søge baglens gennem bufferen. Man kan således bruge `C-f` og `C-S-f` til at søge forlæns og baglæns.

`C-H-f` — *Find reverse*
Finder noget baglæns, se `C-f`

`C-r` — *Replace*
Spørger først om en tekst der skal erstattes, spørger så efter en tekst der skal erstatte den. Nu bevæger Emacs sig fra hvor point er ned over alle matches og spørger om de skal erstattes, hvis man trykker `<space>` eller `y` bliver teksten erstattet, hvis man

trykker `n` går man videre til næste match uden at erstatte det nuværende. Hvis man trykker `!` erstattes alle følgende matches uden at spørge.

`C-z` — *Undo*
Fortryder den seneste ændring, hver opmærksom på at Emacs måde at håndtere undos på er lidt anderledes. Hvis man i et "normalt" program gør handling A, B, og så C, og trykker undo og nu gør handling D, vil historikken huske A, B og D, undo og redo kan således kun komme frem og tilbage mellem disse. Emacs er lidt anderledes, her findes "redo" ikke, tilgengæld er undo en handling der kan undo'es i sig selv. Så hvis man udfører handling A, B og så C, så trykker man undo og så D, så vil undo-historikken se således ud: A, B, C, undo C, D. Man kan så bruge undo til at komme tilbage til C. Tilgengæld vil disse ændringer også være synlige, således kan Emacs huske alt, men det kan godt være lidt uoverskueligt.

`C-t` — *Start shell (terminal)*
Starter en eshell i Emacs.

`C-b` — *Switch buffer*
Kommer op med en menu der lader dig skifte buffer i det nuværende vindue.

`<menu>` — *buffer menu*
Menu knappen er den der sidder i højre side mellem `<AltGr>` og den højre `<Ctrl>` knap. Den ligner typisk en lille menu med en cursor (og findes ikke på Mac). Når man trykker på den åbner den en buffer med en liste af alle åbne buffere i Emacs. Man kan nu bevæge point ned til at pege på en linje og trykke enter for at åbne denne buffer.

`<f5>` — *Close window*
Lukker det nuværende vindue. Vær opmærksom på at hvad du normalt ville kalde et vindue, kalder Emacs et *frame*, mens Emacs bruger termet "window" om dens interne måde at vise flere buffere. Man kan ikke lukke det sidste/eneste vindue på denne måde.

`<f6>` — *Split window horizontal*
Splitter det nuværende vindue så der lægges to ved siden af hinanden.

`<f7>` — *Setup revy windows*
Opsætter vinduerne i Emacs så det er smart i forhold til revyen. Emacs bliver opdelt til ialt to vinduer, det ene er aktoversigten det andet er hvad man ellers arbejder med.

`<f8>` — *other window*
Skifter fokus til det næste vindue.

`C-æ` — *other window*
Skifter fokus til det næste vindue.

`C-ø` — *Balance windows*
"Balancerer" windowerene, så størrelserne bliver ensartede.

`<f1>` — *revy manus insert comment*
Indætter en kommentar i et ubertex dokumment, spørger om hvad du gerne vil have indsat i kommentaren, når du trykker `<enter>` indættes en `\comment{}` med indholdet. Det kan være en smart måde hvis man står og afvikler en sang til en prøve, og man lægger mærke til en fejl, hvis man forsøger at rette den med det samme kommer man muligvis bagud, og hvis man venter glemmer man måske hvor præcis det var. Hvis man skynder sig at trykke `<f1>` og kaster en hurtig kommentar sammen vil

denne blive indsat hvor point er nu, så kan man rette tingene til senere. Hvis man glemmer det så har man dog ikke ødelagt noget ved et eller andet forhastet og forfejlet forsøg.

Note: I Emacs bliver Alt knappen af historiske årsager kaldt Meta

Disse fire genvejstaster gør at du kan holde `<Alt>` nede og så bruge WASD til at flytte point rundt.

`M-w` — *Previous line*
Bevæger point til forrige linje.

`M-s` — *Next line*
Bevæger point til næste linje.

`M-a` — *left char*
Bevæger point et tegn til venstre.

`M-d` — *Right char*
Bevæger point et tegn til venstre.

`<home>` — *revy next*
Avancerer revy cursoren fra hvor den er nu videre til den efterfølgende instruktion eller slide. Hvor pointeren er har ingen indvirkning. Centrerer derudover cursoren midt på skærmen.

`<end>` — *revy enter*
Eksekverere instruktionen eller det slide der er under point. Hvis point er udenfor en instruktion eller slide, vil det første efterfølgende blive eksekveret.

`<delete>` — *revy blank*
“Blanker” skærmen. Se 7.3.2.

C En guide til Linux

cd ls cp mv

ip ssh nano

hvad er /dev/sdX og /dev/sdX1,2,3 osv? hvordan fungerer IP'er?

xmonad nævn

```
xmonad --recompile
xmonad --restart
```

D Installation af Arch Linux

Dette appendix gennemgår installationen af Arch Linux fra bunden. Arch Linux er en såkaldt "Rolling release" distribution af linux, den er ikke specielt begynder venlig, men gør hvad man beder den om, hverken mere eller mindre.

Dette afsnit er ikke et du skal bruge normalt som AV-mand, men dokumenterer processen om at sætte systemet op fra bunden af, normalt vil der forhåbentligt være nogle allerede fungerende installationer du kan låne.

Guiden er baseret på `https://wiki.archlinux.org/index.php/Installation_guide` og
`https://wiki.archlinux.org/index.php/Beginners'_guide`, og antager at du har styr på linux, se evt. Appendix C.

Note: `http://www.muktware.io/arch-linux-guide-the-always-up-to-date-arch-linux-tutorial/`
`http://www.dedoimedo.com/computers/grub-2.html`

Husk at der er "auto completion" på tab-knappen.

D.1 Live USB

Først downloades `.torrent` filen fra `https://www.archlinux.org/download/` og åbnes med dit yndlings torrent program, f.eks. `rtorrent`.

Når `.iso` filen er hentet kan denne brændes til et usbstick. `https://wiki.archlinux.org/index.php/USB_flash_installation_media`

For at lave en live USB fra et allerede eksisterende linux system sættes USB'en i, uden at mounte den (eller unmount den).

Følgende kommando sletter alt på usbsticket og laver et live USB

```
sudo dd bs=4M if=/path/to/archlinux.iso of=/dev/sdX status=progress && sync
```

Hvor `/path/to/archlinux.iso` er stien til `.iso` filen. `/dev/sdX` er stedet hvor USB'en findes, f.eks. /dev/sdb, bemærk at det er uden partitionen, så IKKE `/dev/sdb1`.

Sæt nu USBen i maskinen der skal installeres, og start op fra USB'en.

For at gendanne Live USBsticket til et "normalt" USB stick

```
dd count=1 bs=512 if/dev/zero of=/dev/sdX && sync
cfdisk /dev/sdX
```

Hvis filsystemet Ext4 ønskes:

```
mkfs.ext4 /dev/sdX1
e2label /dev/sdX1 USB_STICK
```

Eller for et klassisk Windows Fat32 system

```
mkfs.vfat -F32 /dev/sdX1
dosfslabel /dev/sdX1 USB_STICK
```

For at lave et Fat32 system kræver det at `dosfstools` er installeret. cfdisk bruges til at lave en partition på USB'en, som findes bå `/dev/sdX1`, `USB_STICK` er navnet der ønskes på USB'sticket.

D.2 Installation af styresystem

Hvis man ønsker et andet tastatur layout det gøres med f.eks. `loadkeys colemak` hvis man ønsker at slå bip-lyden fra kan man gøre det med `rmmod pcspkr`

```
timedatectl status
```

Vær sikker på at tiden er indstillet korrekt, ovenstående skal sige at **Universal time** er korrekt, din lokale tidszone kan indstilles senere. Hvis tiden ikke passer ændres den i BIOS'en inden boot.

Note: Brug ntp? `timedatectl set-ntp true`

D.2.1 Opret forbindelse til internettet

Hvis det er et helt almindeligt trådet netværk virker det muligvis uden problemmer Hvis maskinen er på et seperat netværk uden internet, men med en gateway som brugt til den normalle revyopsætning gøres følgende:

```
ip link show
ip link set enpXsX up
ip addr add 192.168.0.XXX/24 dev enpXsX
ip route add default via 192.168.0.YYY
echo nameserver 8.8.8.8 > /etc/resolv.conf
```

`XXX` er ip'en til maskinen, `YYY` er ip'en til gateway'en

Note: Henvis til gateway

Hvis det er et almindeligt trådløst netværk benyttes `wifi-menu` eller `wpa_supplicant` for mere avancerede opsætninger

Test om der er internet:

```
ping google.com
```

tryk Ctrl-c for at stoppe.

D.2.2 Opret partitioner

Hele situationen omkring UEFI vs. BIOS, GPT vs. MBR, kombineret med bootloaderen (grub) og hardwareunderstøttelse er noget gøjl. Det er et virvar af forskellige systemer der ikke altid har lyst til at samarbejde, ikke fortæller hvorfor det ikke virker, og mangler dokumentation om hvordan man får det til at virke. I en periode, da det hele var nyt, og denne guide blev skrevet i første omgang, var dokumentationen nærmest ikke eksisterende og meget af det her er opdaget ved manuel afprøvning.

De næste par afsnit handler om forskellige måder jeg har prøvet at sætte systemer op på. Oprindeligt brugte jeg GPT-BIOS, da det var hvad jeg endelig fik til at virke, men prøv dig frem.

D.2.3 Opret partitioner (MBR-BIOS)

```
fdisk /dev/sda
```

med `parted -l /dev/sda` kan man se om det er GPT eller ej (også kaldet msdos) tryk `o` for at skifte type til MBR. Lav en swap partition og root + evt. home partition sæt resten op som et GPT-BIOS system

D.2.4 Opret partitioner (GPT-BIOS)

Følgende beskriver hvordan jeg plejer/plejede at oprette partitioner, se det alternative afsnit D.2.5 hvis et UEFI system ønskes. Jeg tivler på at dette er den korrekte måde at gøre det på, men det virker på ældre maskiner.

`lsblk` viser alle partitioner på alle diske, alternativt brug `fdisk -l`

Note: Følgende sletter alt indhold på harddisken og alle eksisterende partitioner

Man kan slette alt indholdet på en disk med

```
sgdisk --zap-all /dev/sdX
```

```
cgdisk /dev/sdX
```

Formententligt `sda`

Tryk enter. (til alt "brok")

`Delete` alle partitoner.

Lav plads til GPT partition *Dette giver plads til en partions tabel til grub*

1. `New`
2. Tryk enter, first sector skal bare være default.

3. `1M` enter - *tidligere 1007KiB*
4. `ef02` enter
5. Tryk enter, der behøver ikke at være et navn

Lav swap *Dette er næppe nødvendigt, men jeg laver den af gammel vane. Alternativt kunne man lave en swap fil, men en partion er simplest*

1. Tryk ned. (til det store område med free space.)
2. `New`
3. Tryk enter, default er fint, formententligt 2048.
4. `3G` mindre swap kan også vælges (eller udelades).
5. `8200` swap Hex koden.
6. tryk enter.

Lav en partition Hvis flere seperate partitioner ønskes, f.eks. opdelt `/` og `/home` oprettes de her.

1. Tryk ned.
2. `New`
3. enter
4. enter
5. enter
6. enter

Vælg `Write`, skriv `yes`, og afslut.

Nu kan der stå at den gamle partitionstabel stadig er i brug. I så fald genstart og udfør alle trinene inden `cgdisk` igen.

Når du er klar skal vi så lave nogle filsystemer. For at få overblik over partitionerne:

```
lsblk
```

```
mkfs.ext4 /dev/sda3
mkswap /dev/sda2
swapon /dev/sda2
```

(Hvis flere partioner blev oprettet til home og root, så gentag øverste linje for disse partioner)

Ignorer `sda1` indtil videre.

```
mount /dev/sda3 /mnt
```

Hvis du skulle have lavet en seperat partition til home så lav en mappe `mkdir /mnt/home` og mount home partitionen der `mount /dev/sdaX /mnt/home`.

D.2.5 (GPT-UEFI)

Dette er en lidt simplere gennemgang af at sætte et GPT-UEFI system op. Dette er den nye måde at gøre tingene på som understøttes af moderne hardware.

`lsblk` viser alle partitioner på alle diske, alternativt brug `fdisk -l`

Note: Følgende sletter alt indhold på harddisken og alle eksisterende partitioner

Man kan slette alt indholdet på en disk med

```
sgdisk --zap-all /dev/sdX
```

```
cgdisk /dev/sdX
```

Formententligt `sda`

Tryk enter. (til alt "brok")

`Delete` alle partitoner.

Lav plads til GPT partition *Dette giver plads til en EFI partitionstabel*

1. `New`
2. Tryk enter, first sector skal bare være default.
3. `512M` enter
4. `ef00` enter
5. Tryk enter, der behøver ikke at være et navn

Lav swap *Dette er næppe nødvendigt, men jeg laver den af gammel vane. Alternativt kunne man lave en swap fil, men en partion er simplest*

1. Tryk ned. (til det store område med free space.)
2. `New`
3. Tryk enter, default er fint, formententligt 2048.
4. `3G` mindre swap kan også vælges (eller udelades).
5. `8200` swap Hex koden.
6. tryk enter.

Lav en partition Hvis flere seperate partitioner ønskes, f.eks. opdelt `/` og `/home` oprettes de her.

1. Tryk ned.

2. `New`
3. enter
4. enter
5. enter
6. enter

Vælg `Write`, skriv `yes`, og afslut.

Nu kan der stå at den gamle partitionstabel stadig er i brug. I så fald genstart og udfør alle trinene inden `cgdisk` igen.

Når du er klar skal vi så lave nogle filsystemer. For at få overblik over partitionerne:

```
lsblk
```

```
mkfs.fat -F32 /dev/sda1
mkfs.ext4 /dev/sda3
mkswap /dev/sda2
swapon /dev/sda2
```

(Hvis flere partioner blev oprettet til home og root, så gentag øverste linje for disse partioner)

```
mount /dev/sda3 /mnt
mkdir /mnt/boot
mount /dev/sda1 /mnt/boot
```

Hvis du skulle have lavet en seperat partition til home så lav en mappe `mkdir /mnt/home` og mount home partitionen der `mount /dev/sdaX /mnt/home`.

Du kan bruge `lsblk` til at få overblik over partitionerne, og hvor de er mounted.

D.2.6 Installation

Sørg for at være på nettet da vi nu skal hente pakker ned til styresystemet.

```
pacstrap -i /mnt base base-devel
```

Tryk enter til spørgsmål.

```
genfstab -U -p /mnt >> /mnt/etc/fstab
arch-chroot /mnt /bin/bash
nano /etc/locale.gen
```

Fjern kommenteringen til linjerne *#da_DK.UTF-8 UTF-8* og*#en_US.UTF-8 UTF-8*

Note: Henvis til brug af nano

```
locale-gen
echo LANG=en_US.UTF-8 > /etc/locale.conf
export LANG=en_US.UTF-8
ln -s /usr/share/zoneinfo/Europe/Copenhagen /etc/localtime
hwclock --systohc --utc
echo XXX > /etc/hostname
```

hvor XXX er navnet til maskinen

Note: Hvis du hellere vil arbejde med at netværksinterfacene hedder `eth0` osv. istedet for enpXsX: `touch /etc/udev/rules.d/80-net-setup-link.rules`

Note: ?

kopieret fra noter: Brug netctl til automatisk at gå på netværk (Ret i hovrdan man bruger deet i resten af dokumentationen)

```
curl "https://raw/githubusercontent.com/Pilen/ubertex/master/linux/revynet" >
 /etc/netctl/revynet"
nano /etc/netctl/revynet
```

Og så skal det gerne matche følgende:

```
Description='Default static network configuration for the AV-setup'
Interface=eth0
Connection=ethernet
IP=static
Address=('192.168.0.XXX/24')
Gateway='192.168.0.YYY'
DNS='8.8.8.8'
SkipNoCarrier=yes
```

Hvor XXX erstates med den lokale IP, og YYY med gateway'ens.

```
netctl enable revynet
```

Hvis man ændrer i configurationen, f.eks. IP'en:

```
netctl reanable revynet
netctl restart revynet
```

Hvis maskinen skal kunne gå på trådløst netværk efterfølgende;

```
pacman -S iw wpa_supplicant dialog
```

```
mkinitcpio -p linux
```

```
passwd
```

Skriv løsn til root (f.eks. `hamsterroot`).

Note: Du kan med fordel oprette din bruger her

Note: Du kan med fordel sætte sudo op her

Note: Du kan med fordel installere sshd her

For et GPT-BIOS system:

```
pacman -S grub
grub-install --target=i386-pc --recheck --debug /dev/sda
grub-mkconfig -o /boot/grub/grub.cfg
```

For et GPT-UEFI system:

```
mkdir /boot/efi
pacman -S grub efibootmgr
grub-install --target=x86_64-efi --efi-directory=esp --bootloader-id=grub
grub-mkconfig -o /boot/grub/grub.cfg
```

Note: Den vil klage og sige: `efibootmgr: EFI variables are not supported on this system`, ignorer dette?

Vi er nu klar til at genstarte op i det nyinstallerede system:

```
exit
umount -R /mnt
shutdown -h now
```

Note: ?

Note: Hvis grub-mkconfig fejler:

```
nano /etc/default/grub
...
...
#fix broken grub.cfg gen
GRUB_DISABLE_SUBMENU = y
```

Hiv usbstikket ud.

Tænd datamaten igen. Hvis du som mig til jubilæumsrevyen installerede systemet på en harddisk i en anden datamat en dens egen, kan det være at ramdisken fejler, i grub vælges da fallback løsningen og du kalder `mkinitcpio -p linux` for at skabe et nyt korrekt image.

D.3 Opsætning

D.3.1 Bruger

Note: Hvis en bruger og sudo er sat op kan man logge ind som dem, ellers så log ind som root

Log ind som root

Note: Kom på nettet igen

```
ip link show
```

For at se netværks interfacet

```
useradd -m -s /bin/bash revy
```

```
passwd revy
```

Giv revy et løsn.

Sudo er allerede installeret via base-devel, du kan læse mere om sudo i Appendix C.

```
visudo
```

Tryk pil ned til du finder linjen

```
root ALL=(ALL) ALL
```

placer cursoren under denne og tryk `i` tast `revy ALL=(ALL) ALL` tryk enter, tryk escape tryk :wq enter.

revy kan nu sudo'e.

```
pacman -S openssh
```

Note: Det er nu default at `PermitRootLogin` er sat til `no`, så det er ikke længere nødvendigt at ændre det i `/etc/ssh/sshd_config`

```
systemctl enable sshd.service
systemctl start sshd
```

Nu kan vi ssh'e ind fra vores lokale maskine `ssh revy@192.168.0.XXX`. *Læs evt. afsnittet om ssh-nøgler.*

Nu kan vi vælge at køre kommandoerne via en ssh forbindelse. Du kan også genstarte og logge ind som almindelig bruger istedet, flere af kommandoerne kræver da sudo.

D.3.2 Grafisk system

Nu skal vi installere et grafisk interface

```
pacman -S xorg-server xorg-server-utils xorg-xinit
```

Den spørger nu hvilken udbyder af `libgl` der ønskes, default (1 mesa-libgl) er formententligt et fint valg (medmindre andet ønskes). (tryk 1 efterfulgt af enter)

Den spørger så om hilken udbyder af `xf86-input-driver` der ønskes, 1: `evdev` eller 2: `libinput`. Libinput er et nyere wayland projekt der også virker til xorg. Begge er tilsyneladende fine, jeg valgte evdev da det er det "klassiske" valg.

Der skal formententligt installeres nogle grafik drivere, læs mere her `https://wiki.archlinux.org/index.php/Xorg#Driver_installation` `fbdev` og `vesa` er nogle udemærkede open source fallback drivere hvis ikke der findes et grafikkort (bemærk at det fører til software rendering). Hvis det er et intel kort, er intel driveren et godt bud, for nvidia er nouveau driverne et godt open source bud

Note: mesa og mesa-libgl er installeret af xorg-server

```
pacman -S xf86-video-fbdev xf86-video-vesa xf86-video-intel
```

Derudover er det en god idé med Hardware video acceleration

```
pacman -S libav-intel-driver libav-mesa-driver
```

Som windowmanager bruger vi xmonad, en simpel "tiling" windowmanager med minimale vinduesdekorationer, kan styres med tastaturet og fordi jeg kender den. Se Appendix C for mere om brugen.

```
pacman -S xmonad xmonad-cotrib
```

Vi skal oprette en `.xinitrc`, den kan hentes sådan

```
curl "https://raw/githubusercontent.com/Pilen/ubertex/master/linux/.xinitrc" > .xinitrc
chown revy:revy .xinitrc
nano .xinitrc
```

Indholdet skal da være:

```
xdotool mousemove 10000 10000                          # Flyt cursor til meget langt væk
xsetroot -cursor .emptycursor.xbm .emptycursor.xbm     # slå cursor fra
xset -b                                                # fjern bell/bip lyd
xset -dpms                                             # Slå energi besparelse for skærmen
xset s off                                             # Slå pauseskærm fra
exec xmonad                                            # Start xmonad/windowmanager
```

Den usynlige cursor hentes med:

```
curl "https://raw/githubusercontent.com/Pilen/ubertex/master/linux/.emptycursor.xbm" > .em
```

```
mkdir .xmonad
curl "https://raw/githubusercontent.com/Pilen/ubertex/master/linux/xmonad.hs" > .xmonad/xm
chown -R revy:revy .xmonad
```

```
import XMonad
import XMonad.Layout.NoBorders

main = xmonad $ defaultConfig {
  terminal = "urxvt",
  borderWidth = 1,
  normalBorderColor  = "#000000",
  focusedBorderColor = "#000000",
  modMask = mod4Mask, -- Set win as mod key
  layoutHook = smartBorders $ layoutHook defaultConfig
  }
```

Note: Skal borderWidth være 0?

```
xmonad --recompile
```

Åben `.bashrc` og tilføj i bunden:

```
if [[ -z $DISPLAY && $(tty) = /dev/tty1 ]]; then
    exec startx
fi
```

Du kan evt. også tilføje følgende alias over linjerne:

```
alias d='export DISPLAY=:0'
```

For at systemet automatisk kan logge ind ved opstart kan man gøre følgende

```
curl "https://raw/githubusercontent.com/Pilen/ubertex/master/linux/autologin.conf" > /etc/s
nano /etc/systemd/system/getty@tty1.service.d/autologin.conf
```

indhold af `autologin.conf`:

```
[Service]
ExecStart=
ExecStart=-/usr/bin/agetty --autologin revy --noclear %I 38400 linux
```

Note: I noterne står det som `curl "https://raw/githubusercontent.com/Pilen/ubertex/master/linux` Outputet fra curl kan skjule forespørgslen efter løsen, så hvis den hænger er det nok derfor

D.3.3 Pakker

```
pacman -S alsa-utils rxvt-unicode dmenu xdotool feh mplayer ttf-dejavu screen htop rsync mu
```

D.4 Master interface

Note: Hvis der ønskes et grafisk interface for at lave en master

```
sudo useradd -m -s /bin/bash master
```

```
sudo passwd master
```

Giv revy et løsn.

```
sudo pacman -S zsh emacs
```

```
ssh-copy-id $WORKER
```

```
sudo pacman -S lxde
```

Bare installer det hele `all`

Automount af USB.

Jeg lavede en fil til polkit, (også til penguin) dette virkede ikke (eller var ihvertfald ikke nok `sudo nano /etc/polkit-1/rules.d/10-enable-mount.rules`

```
polkit.addRule(function(action, subject) {
    if (action.id == "org.freedesktop.udisks2.filesystem-mount-system" && subject.isInGroup
        return polkit.Result.YES;
    }
});
```

Jeg installerede også

```
sudo pacman -S gvfs gvfs-afc
```

Det virker kun med disse to! Hvis man gør begge dele virker det ihvertfald, måske også med mindre.

Dette får også trash til at virke på magisk vis...

Husk at reboote.

D.5 Opdater efter lang tid

Hvis det er virkeligt lang tid siden du har opdateret

```
sudo rm -rf /etc/pacman.d/gnupg/
sudo pacman-key --init
sudo pacman-key --populate archlinux
sudo pacman -S pacman archlinux-keyring
```

E Worker-protokol

Note: Her burde følge en introduction af protokollen mellem master og worker.

Husk at slukke projektorerne når du går!

Det forværer deres holdbarhed at lade dem stå tændt en hel revy-uge.