**Min redovisningssida**

**Kmom01**

Min utvecklingsmiljö består, precis som under de tidigare kurserna, av min MacBook och editorn Sublime Text 2. Jag har valt att installera Debian i en VirtualBox på min laptop, bl.a. för att det är en setup jag hört mycket talas om men aldrig kommit mig för att prova själv - toppen att jag nu fick anledning att skrida till verket! På jobbet är i princip alla servrar vi använder virtuella, men där är det ju inte jag som administrerar dem.

Framöver (d.v.s. så snart GitHub godkänt mig som student) tänker jag även prova på att använda Digital Ocean för ja, det vore bra att ha en aning om hur det går till också. Möjligen skaffar jag också en Raspberry Pi - vi har redan två i hushållet och det lockar att skaffa en egen för labbändamål. Det finns så mycket jag skulle kunna/vilja göra.

Linux har jag haft på min jobbdator i drygt fyra år, men med tanke på att jag mest läser mail och surfar på nätet har det inte gett mig särskilt mycket egentlig erfarenhet av operativsystemet, varken som användare eller systemadministratör. Det var till stor del därför jag valde just det här kurspaketet - jag ville lära mig Linux och mobilt tänkande.

Att installera Debian på den virtuella maskinen var enkelt och jag uppskattade särskilt att VirtualBox hjälpte till genom att läsa av inmatningsfältet för namn på den tilltänkta maskinen och därav förstå att jag tänkte installera Debian samt förvalde det i listan. En liten hake var att VirtualBox per default använde den vänstra kommando-tangenten för att "frigöra" sig från den virtuella servern, vilket hela tiden krockar med mitt skrivande eftersom det är den kommando-tangenten jag använder. Det var dock enkelt att ändra.

Det jag däremot inte fått att fungera (ännu) är copy/paste, så där får jag söka vidare.

Jag såg en del störningsmoment i form av varningsmeddelanden kopplade till locale, vilka försvann när jag definerade LC\_ALL (något man avråds från att göra). Fenomenet uppstod dock inte alls i den minimala Debian-installation jag också gjorde, så kanske använder jag primärt den för kursen; någonstans lockar det att köra maximalt avskalat.

I mitt tidigare yrkesliv har jag jobbat med systemadministration av flera UNIX-varianter och med det i ryggen kändes ingenting konstigt eller obekvämt med vare sig installation eller terminalen, även om jag inte i modern tid gjort någon Linux-installation. Minns dock att jag provade på Mandrake Linux 1999/2000 och att det då var rätt knöligt att få till konfigurationen; att installera Debian idag är en betydligt mer positiv upplevelse.

Det enda i det hela som tog tid, tycker jag, var själva installationen. Väntan blir lång och någonstans mitt i dyker det upp fler frågor så man kan inte lämna den utan uppsikt. Dessutom installerade jag flera gånger, bl.a. på engelska och på svenska, för att se skillnaden. Operativsystem på svenska är obegripligt, så det ska jag fortsätta undvika. Vad då "Hanterar utlösare för systemd"!? (d.v.s. "Processing triggers ..." på engelska) Och slutligen gjorde jag en installation med så lite som möjligt, vilket verkar vara den som funkar bäst - jag har inte behövt göra några konstiga justeringar så här långt.

Jag gjorde en asciinema-inspelning av ren nyfikenhet - bra och användbar funktion!

En allmän kommentar är att instruktionerna för kursmomentet känns lite utspridda och ligger i lager på lager med många länkar till andra ställen; det var svårt att få överblick över vad jag egentligen behövde göra för att bli klar med uppgifterna. De foruminlägg det länkades till var dock föredömligt tydliga och jag har förståelse för att det lätt blir fragmentariskt när mycket inte är kursspecifikt utan används även i andra kurser.

Slutligen står det report.php (i st f .html) på sidan <http://dbwebb.se/linux/redovisa>.

Min me-sida: [http://www.student.bth.se/~masc15/dbwebb-kurser/linux/me](http://www.student.bth.se/%7Emasc15/dbwebb-kurser/linux/me)

**Kmom02**

Det som var helt nytt för mig i detta kursmoment var tmux. Screen kände jag till sedan tidigare (även om jag inte direkt använt det), men tmux var nytt och det känns väldigt användbart när man inte har något grafiskt gränssnitt mot sin server. Eller bara inte vill dra igång multipla terminalfönster. Och för att bevara sessioner mellan inloggningar!

Jag installerade tmux både på Mac:en och på Debian-servern och knappade runt ett tag för att få in känslan. En sak jag snabbt ändrade efter att ha hämtat konfigurationsfilen var prefix-nyckeln - jag ställde om den till Ctrl-B eftersom Ctrl-A krockar med editering av kommandoraden, där Ctrl-A enligt min ryggrad betyder "hoppa till början av raden". Jag upptäckte att detta är ett kommandoradskommando jag använder väldigt ofta.

Jag hade heller inte använt tree-kommandot tidigare, så det var trevligt att lära sig. Eftersom jag använde diff för att jämföra min output med facit (correct.txt) skickade jag sedan också min output vidare via sed-kommandot för att justera antalet filer. Vilka 11 filer hade ni hos er? Hos mig finns det bara 10. Jag skapade tomma filer för de som saknades, och valideringen tvingade mig sedan att skriva in "<!DOCTYPE html>". Efter att ha läst innehållet i main.js replikerade jag innehållet från index.html till genie.html och lade till klassen 'genie' på ett element, för att bilden skulle visas på den sidan.

Det var också nyttigt att få öva sig på att använda rsync - tidigare har jag alltid bett sambon om hjälp med det exakta kommandot när jag behövt skicka saker mellan datorerna här hemma, men det lär inte hända igen för nu vet jag vad jag gör. Och så lärde jag mig att rsync går att köra över ssh, vilket självfallet alltid borde göras mot externa maskiner! När jag jobbade som UNIX-admin använde man rsh/rcp, men jag har lärt mer om säkerhet genom åren och anser det självklart att använda secure shell.

I övrigt känner jag mig fortfarande ganska väl hemma bland ssh-nycklarna och Apache-konfigurationen. Just namnbaserade virtuella hostar har jag inte jobbat så mycket med förut; på min tid skiljde man sina hostar åt främst genom olika portnummer. Emellanåt använde man också olika ip-adresser, men sällan namngivna. Så hur känns konceptet? Bra, rimligtvis - samma server kan hantera många olika webbplatser. Jag visste faktiskt inte heller att man kan definiera variabler i Apache-konfigurationen - även det praktiskt!

Sist några tankar och observationer kring kursmomentets innehåll/instruktioner:

Om sidan [Kom igång med SSH-nycklar](http://dbwebb.se/kunskap/kom-igang-med-ssh-nycklar): Kommandot "ssh-copy-id" finns inte på min Mac, så jag kopierade manuellt nyckeln till servern och lade in den i .ssh/authorized\_keys. Dessvärre säger instruktionen att filen ska heta authorized\_hosts, vilket orsakade viss felsökning - fixa gärna det. Inspelningen som visar proceduren anger dock rätt filnamn.

Det stod i uppgiften "Strukturera filer, kataloger och rättigheter i en webbplats" bl.a. att vi skulle ändra rättigheter, men alla rättigheter verkade vara korrekta från början, så jag behövde inte ändra något. Nu råkar visserligen just jag ha ganska bra koll på fil- och katalogrättigheter i ett UNIX-system, men för nybörjare som behöver ställa i ordning webbplatser vore det nog bra att veta i alla fall lite om hur rättighetssystemet fungerar. Det vore bra om det togs upp i kursen, tänker jag - åtminstone översiktligt.

Kommandot för att rsync:a från example till kmom02 (rsync -av example/mySite/ me/kmom02/mysite/) ger inte det resultat som instruktionen uppger. Är detta månne gjort med avsikt för att vi studenter ska lära oss mer genom att tvingas felsöka? :-)

Det står också att adressen man ska ange i webbläsaren för att komma åt sin webbplats är http://mysite.linux.se/, men om det menas att man ska göra det från sin värddator (snarare än på Debian-servern) borde portnummer 8080 också anges. Testar ni oss?

Kanske kursen även kunde nämna scp och sftp, ifall man inte har rsync i andra änden.

Och för oss som gjort sånt här tidigare vore det skoj med extrauppgifter, kanske med lite fler användbara UNIX-kommandon eller något praktiskt systemadministrativt? Konfigurera sin webbplats med säkerhet, SSL/TLS, certifikat etc? Versionshantering? Git är ju ett bra verktyg för många ändamål, och vanligt i jobbsammanhang också numera. Fast en del av sådant kanske kommer senare i kursen, förstås - jag läser inte i förväg.

Min me-sida: [http://www.student.bth.se/~masc15/dbwebb-kurser/linux/me](http://www.student.bth.se/%7Emasc15/dbwebb-kurser/linux/me)

**Kmom03**

Jag har ganska många rader skriptprogrammering under bältet sedan tidigare, och har ju också arbetat som sysadmin med några olika UNIX-system, så uppgifterna i det här kursmomentet bjöd mestadels inte på något nytt. Det halvt-om-halvt nya är Bash då mina tidigare erfarenheter främst kommer från Bourne, Korn och C shell. För mig känns Bash ganska mycket som en upphottad version av Ksh och jag blir glad och inspirerad. Det var synd att Bash inte fanns där och när jag skriptade i princip dagligdags.

En sak jag finner roande med att läsa dessa kurser är att även när jag tror att jag redan kan "allt" så lär jag mig ändå något nytt. Ibland mer, ibland mindre, men alltid något.

Saker jag lärde mig i det här kursmomentet innefattar bland annat att på Mac:en har inte wc-kommandot någon "--lines"-option och man kan (på Debian) skriva "--word" eller "--words". Också nytt för mig var "-L" (Debian, ger längden på längsta raden).

Och kommandot wc -c är inte samma som wc --chars (wc --bytes respektive wc -m), så det får man kanske se upp med. Skillnaden mellan bytes och bokstäver måste man som programmerare ha koll på.

Nya optioner till grep som jag lärde mig under övningen är: "-c", "-m", "-A", "-B", "-C". Det är sådant jag definitivt kommer att ha nytta av på jobbet där servrarna är UNIX:ar.

Krav 9 i hitta-saker-i-en-loggfil-uppgiften anger "forum", men för att hitta något behöver man antingen söka på "Forum" eller söka okänsligt för versaler/gemener. Jag vet inte vilken variant ni tänkte er, men jag sökte okänsligt. Om jag varit sedvanligt bokstavlig här hade jag inte fått några träffar. Kanske värt att tänka på gällande instruktionerna?

Krav 10 är inte heller helt glasklart beskrivet, tycker jag. Vilka rader ska räknas - alla eller bara de där Bobbzorzen skrev något? Jag valde att tolka det som alla rader.

Skoj var det, förresten, att hitta sig själv i chattloggen. Två gånger. En gång p.g.a. mig själv och en gång omnämnd av sambon (som chattade aktivt på dbwebb just då).

Skriptprogrammering är enormt användbart, det vet jag ju inte minst för att jag upplevt och upplever det på jobbet varje dag. Utmaningarna jag ser är nog främst att det man har att tillgå på olika plattformar och operativsystem varierar en del, så det kan vara bra att vara medveten om skillnader och försöka skriva så allmängiltig kod som det går. Och alltid kontrollera argument som användaren anger. Användare (även du & jag!) gör fel.

Jag tittade på ett 10-tal av videosnuttarna och tycker att de verkar bra, främst för den som är nybörjare. Icke desto mindre lärde även jag mig nya saker som t.ex. echo -e, som förstås inte finns på Mac:en. Frågan om extrauppgift förstår jag inte riktigt - det fanns väl inget mer att göra än läsa (och ev. labba själv)? Ja, jag läste igenom artikeln. Antalet poäng borde enligt min beräkning bli 22, då jag ju löste samtliga deluppgifter.

Och så gjorde jag kursmomentet lite mer utmanande genom att faktiskt läsa igenom hela Bash-referensmanualen från början till slut. Det känns bra att skaffa sig en rejäl teoretisk grund, tycker jag, som också är en sådan som gillar att läsa mig till kunskap.

Jag bör nog också nämna att jag tolkade skript-uppgiften mer bokstavligt än ni nog hade tänkt er. Står det att man ska skriva ut en text som innehåller backslash (hello.bash) eller ett dollartecken följt av bokstäver (myFunction.bash, där ni säkert tänkt att det var användarens userid som skulle skrivas ut på skärmen) så gör jag det. Dessutom valde jag (argument\_2.bash) att skriva ut texten "Missing argument" inte bara när man angett för få utan även när man angett för många argument. Hoppas ni står ut med min humor!

Mest utmanande var nog att jag och valideringen hade delvis olika åsikter om hur man kan/ska/bör skriva, men där är jag kanske onödigt pragmatisk och tänker att det funkar ju så här. Ska man skriva skript "på riktigt" krävs mer ordning och reda och framför allt viss paranoia, så validering är ändå något bra. Det är positivt att ni uppfostrar oss!

Min me-sida: [http://www.student.bth.se/~masc15/dbwebb-kurser/linux/me](http://www.student.bth.se/%7Emasc15/dbwebb-kurser/linux/me)

**Kmom04**

Det var riktigt roligt att få prova på node.js - tänk att det kan vara så snabbt och enkelt att göra en egen liten simpel webbserver! Server-koden var den roligaste att skriva och jag associerade ofta till den kod jag skrev i Python-kursen - map, reduce och filter t.ex. Jag måste erkänna att jag blir alltmer förtjust i JavaScript vartefter jag lär mig mer och det känns bra att veta att även jobbet framöver kommer att innehålla mer av språket. Det är ännu en bra sak med att jag sökte och går det här webbutvecklingskurspaketet.

Jag körde snabbt igenom alla övningarna i javascripting-workshopen, och det var bra att fräscha upp minnet då det kändes som om jag hunnit glömma en del av javascriptandet. Sedan körde jag vidare med learnyounode och extrauppgiften i form av några inledande övningar, men fastnade ett tag på insamlingen av resultat från flera asynkrona anrop och avbröt sedan övningen när jag insåg att jag inte behövde köra igenom dem alla.

Värt att nämna är att node-versionerna är väldigt olika mellan Debian (0.10.29) och MacOS (5.8). Det tog mig alldeles för lång tid att fatta att node på Debian måste köras med babel-node för att funka bra - inte ens något så enkelt som string.endsWith(..) fanns att tillgå annars. Detta i sin tur innebar att workshopen learnyounode blev i princip okörbar på Debian, varför jag flyttade över den delen av övandet till min Mac-laptop. Ytterligare en av anledningarna till att jag är nöjd med att vara Mac-användare.

När det så blev dags att koda på riktigt kände jag mig, tack vare övningarna, väl rustad. Det var inga direkta konstigheter som dök upp och implementeringen gick rätt smidigt.

För servern valde jag att definiera en default-port (1337), som används om variabeln LINUX\_PORT inte är satt. Servern startar alltså vare sig man definierat variabeln eller inte. För klienten gjorde jag tvärtom: avslutar programmet med ett felmeddelande om någon av variablerna LINUX\_SERVER eller LINUX\_PORT inte är satt. Omväxling förnöjer, även om jag i professionella sammanhang möjligtvis hade varit mer konsekvent.

Klienten läser även av en variabel, DEBUG, som aktiverar debug-läge i skriptet, vilket innebär att det skrivs ut vilken server och port som används och vilka curl-kommandon som faktiskt körs. Praktiskt vid utveckling, möjligen inte lämpligt för drift i produktion.

Det förvånade mig att dbwebb:s validering av bash-skript var så fantastiskt kinkig, men jag tycker ändå att påpekandena var genomgående bra och på sin plats - det är ju rätt viktigt att tänka på "kvotering" (heter väl citering på svenska, antar jag), och hur olika variabelstrukturer expanderas, just vid shell-programmering. Vi får nyttiga lärdomar!

För att svara på de specifika frågorna: nej, jag har inte programmerat JavaScript på serversidan tidigare. ECMA6 gillar jag - spontant tänker jag genast på möjligheten att använda konstanter och block-scope på variabler. Map/Set etc är också väldigt trevliga nyheter. Haken med mycket av det nya (för mig) i ES6 är att det riskerar att göra JS-kodandet lite väl likt min vardags-Java. Jag tror jag har nämnt risk för sammanblandning förut. Huvudsakligen är ES6 bara positivt! Vad gäller babel-node kan man väl nästan inte vara annat än positiv, då den möjliggör att man använder nya saker även om node inte stödjer dem (ännu)? Nodeschools övningar var okej, men jag hade uppskattat om man kunde få ledtrådar för att hitta rätt när man körde fast - ett "tipsläge". Koncepten kring klient och server är välbekanta för mig sedan länge. Eller missförstår jag månne frågan?

Några kommentarer kring instruktionerna för kursmomentet:

Jag tog mig friheten att låta min server svara med texten “The resource does not exist.” (i stället för "exists", som det står i instruktionen) när den får frågan om en resurs som inte finns. Språk och formuleringar är viktigt för mig, så den kunde jag inte motstå.

"Här skall du även visa innehållet i header-delen av svaret (curl -I)." anger instruktionen för den sista uppgiften, men skriver man så får man ju bara header-delen. Jag använde mig av --include (-i) i stället, för att få med både headers och innehåll.

Jag tyckte att det här var ett roligt kursmoment, med rätt lagom stor omfattning. Det ger en bra introduktion till både klient- och serverprogrammering i webbsammanhang.

Min me-sida: [http://www.student.bth.se/~masc15/dbwebb-kurser/linux/me](http://www.student.bth.se/%7Emasc15/dbwebb-kurser/linux/me)

**Kmom05**

JSON API har jag inte träffat på tidigare och det är bra att veta att specifikationen finns. Jag tycker att det är viktigt med ordning och att ha struktur och gärna specifikationer att vägledas av när man skapar eller använder gränssnitt mot andra. Den kan vara till nytta.

Att skriva skript är i sig inget nytt, men jag skriver sällan skript i den här storleken och ska jag vara ärlig var det inget nöje. Jag saknade ett riktigt programmeringsspråk, med split och join och tänk om jag hade haft en JSON-tolk i stället för att behöva jobba med komma-separerad data! Mest frustrerande var det nog att försöka utvinna matnyttig information ur svaret från servern och jag lade alldeles för mycket tid på att skriva en egen split-funktion innan jag upptäckte att read kan läsa in csv-data till en array.

Jag hade god nytta av att bash har associativa arrayer, även om jag blev tvungen att installera en nyare version av bash på min Mac för att kunna använda dem - Mac:en hade version 3.2 som saknar stöd för bl.a. associativa arrayer, så jag fick installera en nyare version och ändra shebang-raden. Men därefter funkade det alldeles utmärkt.

En annan frustration var det här med att man inte returnerar data från funktioner, vilket ställde krav på mer ovana lösningar. Jag är dock rätt nöjd med att inte behöva ha särskilt många globala variabler. Rumsinformationen lagrade jag i en global associativ array för att enkelt kunna uppdatera och komma åt den inifrån olika funktioner. Namnen på tillgängliga kartor behövdes också på några olika ställen, men i övrigt var det främst konfiguration som blev global för att göra skriptet enkelt att underhålla i framtiden.

Serverkoden blev jag tvungen att titta på för att begripa att det var kartans namn och inte dess id som skulle bifogas vid "välja karta"-anropet och för att se vilka anrop som stöddes använde jag helt enkelt "grep router.get". Ett tag tänkte jag ge mig på att felsöka server-koden eftersom den ibland meddelade att jag valt en ogiltig riktning trots att så inte var fallet, men bedömde snart att det inte var värt besväret. Kul att koden var så pass lik övningen vi gjorde i tidigare kursmoment! Jag noterade också att servern håller spelet i minnet, vilket gör att det efter en krasch inte finns kvar. Det gick för övrigt alldeles utmärkt att få servern att krascha, även om jag nu inte minns exakt hur.

Jag är ganska nöjd med mitt skript, som blev prydligt med ett antal separata funktioner att använda och återanvända vid behov och det blev då också väldigt enkelt att göra del 2 av uppgiften - förvånande lite behövde ändras för att få loopandet att samspela med befintlig kod. Framförallt är jag nöjd med att jag fick prova på, och lära mig, ett antal mer eller mindre avancerade bash-funktioner. Skriptet blev lite av en provkarta, känns det som. Strängmatchningar med "=~" och/eller jokertecken visste jag inte ens fanns (trots att jag har läst hela manualen). Aritmetisk expansion var också användbart och jag fick konkret nytta av att använda returkoden från min "välja karta"-funktion.

Och precis som i det förra kursmomentet hade valideringen en oherrans massa åsikter om min kod, men den hade ju helt rätt och hjälpte mig faktiskt att hitta en bugg där jag missat dollartecknet framför ett variabelnamn. Värdet av verktyg för validering och som guidehjälp är något jag fått upp ögonen för under kurserna i kurspaketet. Kudos!

Det här att skriptet utvecklades från början utan en interaktiv loop var smart för att det gjorde att man inte frestades spara tillstånd i variabler i scriptet utan i stället jobbade tillståndslöst. Rent, ja REST-igt, rentav. Bara spelid och rumsid lagrat i varsin liten fil.

Om kursinstruktionen: det är exakt samma asciinema (1voz3ecbgsbu5dytp9sz5n2kb.js) som visas sist i Del 1 och därefter i Del 2 - det tycker jag borde åtgärdas eftersom dessa små snuttar kompletterar texten bra så att man förstår vad uppgiften går ut på.

Roligt kursmoment som jag lade lite för många timmar på - men jag lärde mig mycket!

Min me-sida: [http://www.student.bth.se/~masc15/dbwebb-kurser/linux/me](http://www.student.bth.se/%7Emasc15/dbwebb-kurser/linux/me)

**Kmom06**

Det här kursmomentet gick ganska bra, även om jag dock inte riktigt förstod varför instruktionen tyckte att jag skulle lägga till en metod (sic!) vid namn placeRandom() i GomokuServer.js - det borde ju räcka utmärkt med att bara definiera en ny route?

I vilken enhet mäter vi hur väl jag fått ordning på begreppen klient och server? Alltså, jag anser mig veta vad en klient resp. en server är och hur de fungerar ihop, men ska väl också erkänna att det lätt blir lite förvirrande med färdig kod (som tar tid att lära känna!) och både klient och server i samma språk. Skönt att bara skriva Node.js, dock. Till skillnad från i tidigare kursmoment kom min kod och validatorn riktigt bra överens.

Den asynkrona programmeringsmodellen känns vid första ögonkastet ovan och konstig, men den går väldigt snabbt att vänja sig vid och jag har nu nästan svårt att tänka mig något annat sätt att programmera. Icke desto mindre krävs förstås bra mycket mer kontinuerlig användning för att känna sig helt trygg och bombsäker på hur det funkar.

Kommandoradsklienterna i Node.js gick det bra med, men jag funderade en del på det här med när det är lämpligt att använda sig av console.log kontra att skriva direkt mot stdout. Eller det här med att kasta fel i stället för att direkt skriva ut felmeddelanden.

Det enda jag egentligen gjorde med grundkoden till spelet var att jag kompletterade felmeddelandet om storlek på spelplanen så att man får reda på vilket det godkända storleksintervallet är (i GomokuBoard.js) - det är ju onödigt och frustrerande att få veta att man skrivit fel men inte få information om hur man ska bete sig för att göra rätt. Utöver det lät jag det mesta vara orört - lade bara till de olika random-funktionerna.

Och som svar på den raka frågan: nej, jag känner mig inte redo för någon Gomocup och tur var väl det med tanke på att årets version avslutades idag. I ärlighetens namn skulle det krävas rätt mycket mer arbete för att få ihop något som skulle vara redo för tävling. Den här gången utförde jag p.g.a. tidsnöd dessutom heller inga av extrauppgifterna.

Min me-sida: [http://www.student.bth.se/~masc15/dbwebb-kurser/linux/me](http://www.student.bth.se/%7Emasc15/dbwebb-kurser/linux/me)

**Kmom07/10**

**Krav 1 Bash för att konvertera XLS till JSON**

Jag öppnade Excel-filen med hjälp av Libre office och sparade dess innehåll komma-separerat - utan citattecken för att skilja fälten åt, eftersom jag ville göra det enkelt att i Bash-skriptet identifiera eventuella tomma fält. Sedan tittade jag på datat för att se vilka konstigheter jag skulle behöva ta hänsyn till i kodandet. Särskilt tittade jag efter tomma fält och fält innehållande potentiellt besvärliga tecken. Det visade sig att datat hade tomma fält, både först och sist och mitt i raderna, men dock inga med kommatecken. Skriptet behövde alltså hantera tomma fält, men inte ev. fältavskiljare i fältvärden.

En liten klurighet är att den första raden innehåller en rubrik som är irrelevant för uppgiften, så hur kan skriptet lista ut vilka rader det ska bry sig om? Ett alternativ vore att manuellt radera den första raden i indatafilen och därmed slippa hantera detta fall, men med en sådan lösning måste man komma ihåg att göra detta för varje ny export. Jag valde i stället att ignorera all data fram till den rad som börjar med "Salsnr", från den extrahera kolumn-rubrikerna och därefter betrakta resten av filen som salsdata.

När jag fick svårt att bestämma mig för vilken stil min kod skulle ha och hur jag skulle namnge funktioner och variabler hittade jag Googles style guide för shell scripts och anpassade skriptet efter den. Referens: <https://google.github.io/styleguide/shell.xml>.

Jag lekte även med fildeskriptorerna, mest för att det var kul att prova, och ändrade (efter att ha sparat undan en kopia!) stdout så att den skriver till fil utan att uttryckligen behöva skriva det i koden. På samma sätt ändrade jag stdin så att indata läses från fil i stället för från tangentbordet. I slutet av skriptet återställer jag stdout från kopian för att skriva på skärmen igen. Jo, jag vet att jag skulle kunna skriva på skärmen via kopian, utan att behöva återställa stdout, men då hade jag inte fått testköra flytta-kommandot.

**Krav 2 Server för att visa och söka bland salar**

Här blev det lite jobbigare, att komma ihåg hur man behöver knyta ihop de olika filerna. Jag landade på en lösning med en index.js, för kommandoradsargument, som använder en server-modul, i server.js, som i sin tur använder router.js och klassen BRoomServer, i BRoomServer.js. BRoom står för BTH Room Application (som kan förkortas "BRA").

Min index.js är förvillande lik index-filen i föregående kursmoment och även router.js återanvänds här tämligen oförändrad. För server.js använde jag GomokuServer.js som utgångspunkt, men städade upp och anpassade koden enligt behoven för denna uppgift.

BRoom-klassen tar via konstruktorn emot namnet på JSON-filen som innehåller salsdata, och läser därefter in detta så att vi är redo att börja jobba så snart servern har startat. Om något fel uppstår vid inläsningen kastas en exception, vilket hindrar servern från att starta, och ett felmeddelande skrivs ut i konsolen. Ett smart trick vid inläsningen är att eliminera den översta objektnivån så att salarna lagras i BRoom-instansen som en lista. Och att hantera listor har vi ju övat ordentligt på tidigare, så där känner jag mig trygg.

Sökning efter alla salar, en specifik sal, alla salar i ett hus eller salar som matchar en söksträng görs på samma sätt: med hjälp av findRooms-metoden i BRoom-instansen, som tar emot ett filter specifikt utformat för sökningen och returnerar en filtrerad lista. När inga salar hittas returneras sålunda en tom lista som servern får välja att hantera.

Filtret skapas av serverns egen findRoom-funktion baserat på sökterm, textjämförare och attributväljare. Varje enskild route anropar funktionen med sina egna parametrar och när ingen filtrering önskas lämnas parameterlistan tom. Här påmindes jag om att tecken utanför ASCII-skalan URI-kodas och därmed behöver avkodas före användning.

All textmatchning görs, för användarvänlighet, oberoende av versaler och gemener. Instruktionen anger inte om hela eller bara delar av salsobjekten ska returneras, så jag har valt att låta servern returnera allt och överlåta till klienterna att göra egen filtrering. Servern förpackar också salslistan under attributet "rooms" i det returnerade objektet, detta för att underlätta framtida utökningar där servern ska leverera annan sorts data.

Jag är inte alldeles säker på att lösningen är optimal, med en enda flexibel sökfunktion, men den funkar. I tjänsten skulle jag nog provköra så här ett tag för se hur det känns.

**Krav 3 Klient i Node.js för att testköra mot servern**

Jag utgick även här från exempel i kmom06, närmare bestämt index-filen för Gomoku-klienten, och justerade efter behov. Min klient består av en klass, BRoomClient, som finns i BRoomClient.js och en index.js, som gör det mesta av jobbet. Klient-klassen sköter kommunikationen med servern och innehåller främst de olika routes som stöds.

Att implementera funktionaliteten för att hämta data från servern enligt menyvalen var enkelt och gick snabbt - det blev en generisk lösning med anrop till önskad operation i klientklassen och därefter skickas resultatet över till en tolka-och-skriv ut-funktion. För att välja vilka attribut som ska presenteras skickas det in en lista med attributnamn. Skulle listan händelsevis saknas eller vara tom inkluderas samtliga tillgängliga attribut.

Vid allmän listning av salar har jag valt att inkludera endast salsnummer, salsnamn, ort, typ och kapacitet (storlek), allt enligt principen att dessa borde vara de mest relevanta. Vid visning av en enskild sal inkluderas samtliga attribut, liksom även vid fritextsökning över samtliga fält. Beträffande sökning av salar per hus anger instruktionen salsnamn så det är vad som visas, vilket också innebär att man inte får se de salar som saknar namn. Skulle man absolut vilja få fram även dessa kan man fritextsöka på husets benämning.

Det som tog mest tid var att få till en acceptabel presentation av salsinformationen. Jag ägnade eoner av tid åt att tänka på hur jag skulle kunna lägga upp det, testa lösningar, skriva kod, strukturera om och slutligen städa bort det mesta av den skrivna koden.

En funktion som blev kvar är en utfyllnads-funktion, pad, i String-prototypen, så att jag kan anropa metoden och få tillbaka strängens värde utfyllt (med önskat tecken, eller ' ') till angiven längd. Den används vid utskrift av fältnamn och -innehåll för att få tydliga kolumner. Texten blir högerjusterad för numeriska värden, annars vänsterjusterad.

Och för att veta hur stora kolumnerna ska vara behöver jag förstås ta reda på hur långa de längsta värdena är och reservera plats för dessa. Därför skrev jag sålunda en funktion som går igenom och analyserar den erhållna salsinformationen och returnerar ett objekt med längden på det längsta värde som hittades för de respektive önskade attributen.

I övrigt har jag bara en funktion, parseAndPrint, som tolkar JSON-datat och omvandlar det till en lista med salar, löper igenom listan och slår ihop alla enskilda (blankutfyllda) attributvärden till en hel rad per salspost och skriver därefter ut salsraderna - efter att först ha skrivit ut en rad med attributnamnen (dock endast då fler än ett visas). Raderna skrivs på stdout eftersom det med console.log inte gick att omdirigera utskriften till fil och jag ville kunna jämföra resultatet från före och efter större uppdateringar av koden.

**Krav 4: Bättre sökalgoritm (optionell)**

För att implementera sökalgoritmen borde det inte behövas mer än en ny route och lite extra logik någonstans i server-koden. Eftersom jag valt att hålla min BRoomServer-klass enkel skulle jag nog välja att skapa en poängsättningsfunktion i server.js. Denna skulle räkna (och spara, på objektet) poängsumman för varje sal, givet valt sökbegrepp. Därefter sorteras listan med salar baserat på poängsumma, med högsta summan först. Detta borde vara enkelt med Array.sort(..) som kan ta emot en sorteringsfunktion.

**Krav 5: Filtrera antalet träffar (optionell)**

Klienten får ytterligare en (optionell) parameter till kommandona list, view och search. Denna parameter omvandlas till en query-parameter i anropet mot servern, som utökas med hantering av parametern och returnerar de max X första salarna från resultatet. Här skulle jag, utöver det som anges i uppgiften, implementera min antalsfiltrering även för den förbättrade sökalgoritmen, som ju alltid ger de mest relevanta träffarna först i listan.

**Krav 6: Bättre testmöjligheter (optionell)**

På grund av tidsbrist utförde jag inte den här uppgiften, men det ser inte ut som om den skulle vara så knepig - den verkar ganska rättfram. Debug-utskrifter och -konfiguration är sådant jag normalt använder under utvecklingsarbetet (men därefter städar bort), så det borde inte medföra några konstigheter att lägga till kommandoradsargument och ett par extra utskrifter via console.log. Bash-skriptet borde gå bra att skriva då jag känner mig hemma i bash numera, så utmaningen är nog att hitta på testfallen. Och jag skulle försöka göra något i enhetstestväg, som rapporterar OK eller FAIL för respektive testfall.

**Allmänt om projektet**

Projektet har varit roligt och bjudit på ganska stor personlig frihet - jag uppskattar att instruktionen varit "fluffig" så att jag fått välja vad jag själv tycker är rimliga vägar. Mot bakgrund av tidigare kursmoment och övningar kändes projektet ganska lätt och gick snabbt, men som nämnts tog det väldigt lång tid att landa i en lösning för presentation och tiden medger tyvärr inte att jag implementerar alla optionella krav denna gång.

**Allmänt om kursen, materialet och handledningen**

Jag tycker att den här kursen varit väldigt rolig och lärorik samt att materialet varit bra. Att utgå från exempel fungerar bra och man kommer igång väldigt snabbt, men ibland kanske det vore bra att tvingas tänka efter mer för att fästa kunskaperna mer på djupet. Kurs- och referenslitteraturen har jag skaffat, men faktiskt knappt använt alls - Internet har funkat bra för mig. Handledningen har jag i vanlig ordning inte nyttjat, men vad jag kan se har lärare och administratörer varit aktiva och svarat snabbt på kursens forum.

Förslag på förbättringar? Ja, utöver vad jag skrivit i tidigare redovisningstexter vore det bra att få in test och avlusning av kod. Hur kan man t.ex. jobba testdrivet och skriva enhetstester för node-applikationer? Sådant gör jag på jobbet och önskar även andra. Inte minst vill jag att alla mina framtida kollegor har sånt med sig redan från början.

Jag är nöjd med kursen och har lärt mig mycket längs vägen. Mitt betyg blir 9 (av 10) och jag skulle gärna rekommendera kursen till andra, men den ges väl inte fristående?

Min me-sida: [http://www.student.bth.se/~masc15/dbwebb-kurser/linux/me](http://www.student.bth.se/%7Emasc15/dbwebb-kurser/linux/me)