Ruby-kurs: Videre.

Kent Dahl < kentda@pvv.org > Linpro utgave v0.3, Mai 11., 2004

Avslutning av Ruby-kurset.

1 Dagens agenda

Ta for oss løst og fast som er aktuelt for å bruke Ruby i en større sammenheng. Testing (for å kunne stole på koden vår), GUI (for å kunne gi programmer til kunde), C/C++ integrering (for hastighet), samt litt om design (for å kunne skrive store programmer).

2 Unit testing

Enhetstesting (eng: *unit testing*) er en testmetode hvor enkeltkomponenter testes hver for seg. Tanken er at når alle enkeltdelene er riktige, vil summen av delene også være riktig. Noen utviklingsmetodologier, slik som "eXtreme Programming" (XP), gjør utstrakt bruk av enhetstesting til å kvalitetssikre programvaren og utviklingsprosessen.

2.1 Test::Unit-rammeverket

2.1.1 Trekant

La oss anta at vi har en metode som regner ut arealet av en trekant.

```
[ code/trekant.rb ]
 1| def trekant_areal( grunnlinje, hoyde )
 2|   grunnlinje * hoyde / 2
 3| end
```

Den enkleste måten å teste denne metoden vil være å kalle den. Så kan vi sammenligne forventet svar med faktisk svar.

```
[ code/test_trekant1.rb ]
 1|     assert_equal( 25, trekant_areal( 10, 5) )
```

Her bruker vi en metode som gjør sammenligningen for oss og gir oss en beskrivende tilbakemelding dersom resultatet fra metoden ikke stemmer med det vi forventer.

`assert_equal`-metoden er definert i biblioteket for enhetstesting så vi må pakke testen vår inn i et `TestCase`.

```
[ code/test_trekant2.rb ]
 1| require 'trekant'
 2| require 'test/unit'
 3| class TestTrekant < Test::Unit::TestCase
 4|   def test_trekant_areal
 5|     assert_equal( 25,  trekant_areal( 10, 5) )
 6|     assert_equal( 7.5, trekant_areal(  5, 3) )
 7|   end
 8| end
```

Når vi kjører denne koden får vi en oversikt som viser resultatet av testen. Hvor mange tester som ble kjørt, hvor mange kall til `assert_*`-metoder som ble gjort, hvor mange av disse **feilet** og om det eventuelt var noen uventede problemer.

Var du litt paff over at det ikke er noe kode som starter kjøringen av testene? De defineres jo bare som metoder i en klasse.

Rammeverket for enhetstesting gjør en del magiske triks ved hjelp av refleksjon. Den ser over alle klasser som arver fra `TestCase`, lager instanser og kjører alle metodene med navn som begynner på `test`.

2.1.2 En samling tester

La oss ta et større eksempel hvor vi har laget en liten klasse `Vector` som representerer tredimensjonale vektorer.

```
[ code/vector.rb ]
 1| # En klasse for tredimensjonal vektorer.
 2| class Vector
 3|   attr_reader :x, :y, :z
 4|   def initialize( x, y, z )
 5|     @x, @y, @z = x, y, z
 6|   end
 7|   def +(other)
 8|     self.class.new( @x+other.x, @y+other.y, @z+other.z )
 9|   end
10|   def -(other)
11|     self.class.new( @x-other.x, @y-other.y, @z-other.z )
12|   end
13|   def *(other)
14|     self.class.new( @x*other.x, @y*other.y, @z*other.z )
15|   end
16|   def to_a
17|     [@x, @y, @z]
18|   end
19| end
```

Når vi så begynner å bruke denne klassen, så må det jo testes slik at vi kan være trygge på at den gjør det den skal. Vi skriver da et test-tilfelle som sjekker alle offentlige metoder i klassen.

[code/test_vector.rb]

```
 1| require 'vector'
 2| require 'test/unit'
 3|
 4| class TestVector < Test::Unit::TestCase
 5|   def setup
 6|     @v1 = Vector.new(1,2,3)
 7|     @v2 = Vector.new(2,3,6)
 8|   end
 9|   def test_accessor
10|     assert_equal( 1, @v1.x )
11|     assert_equal( 2, @v1.y )
12|     assert_equal( 3, @v1.z )
13|   end
14|   def test_to_array
15|     assert_equal( [1,2,3], @v1.to_a )
16|     assert_equal( [2,3,6], @v2.to_a )
17|   end
18|   def test_plus
19|     v = @v1 + @v2
20|     assert_equal( [3,5,9], v.to_a )
21|   end
22|   def test_minus
23|     v = @v2 - @v1
24|     assert_equal( [1,1,3], v.to_a )
25|   end
26|   def test_dot
27|     v = @v1 * @v2
28|     assert_equal( [2,6,18], v.to_a )
29|   end
30| end
```

Disse testene kan da legges sammen med testene for andre klasser og moduler. Da får vi *ett* testprogram, med *ett* resultat som må sjekkes, uansett hvor stort systemet vi utvikler har blitt og hvor mange tester der er.

2.2 Svartboks- og hvitbokstesting

Svartbokstesting (eng: *black box testing*) er testing hvor man ikke bekymrer seg om hva som foregår inni enheten som testes. Man bekymrer seg bare om innput og utput, det offentlige grensesnittet.

Hvitbokstesting tar hensyn til det som foregår inni enheten som testes. Man bruker hvitbokstesting når man f.eks. forsøker å teste alle utføringsstiene (eng: *execution paths*) til enheten.

2.2.1 Ruby og hvitbokstesting

Gitt at vi har en liten klasse som søker etter forekomster av tekst i en større streng.

[code/hvitboks.rb]

```
 1| class Finn
 2|   def initialize( tekst )
 3|     @tekst = tekst
 4|     @indeks = 0
 5|   end
 6|   def neste( soeketekst )
 7|     posisjon = @tekst.index( soeketekst, @indeks )
 8|     @indeks = if posisjon then posisjon + soeketekst.size else @indeks end
 9|     posisjon
10|   end
11| end
```

Den interne indeksen angir hvor neste søk skal fortsette fra, som jo blir første bokstav etter den vi har funnet.

[code/test_hvitboks.rb]

```
 1| require 'hvitboks'
 2| require 'test/unit'
 3|
 4| class TestFinn < Test::Unit::TestCase
 5|   def setup
 6|     @finn = Finn.new("Å være eller ikke være...")
 7|   end
 8|   def test_neste
 9|     assert_equal(  2, @finn.neste( "være" ) )
10|     assert_equal(  6, hent_intern_indeks    )
11|     assert_equal( 18, @finn.neste( "være" ) )
12|     assert_equal( 22, hent_intern_indeks    )
13|     assert_nil( @finn.neste("e") )
14|   end
15|   def hent_intern_indeks
16|     @finn.instance_eval do
17|       @indeks
18|     end
19|   end
20| end
```

3 Grafisk brukergrensesnitt

Masse valgmuligheter. Tk, *FOX*[1], *wxWidgets*[2], Qt, *GTK*[3], Motif, *FLTK*[4] og stort sett de fleste andre GUI-bibliotekene har Ruby-bindinger.

3.1 FOX

FXRuby[5] er en utvidelse for å kunne bruke det grafiske brukergrensesnittet *FOX* i Ruby.

[code/fxheiverden1.rb]

```
 1| require 'fox'
 2|
 3| class Verden
 4|   include Fox
 5|
 6|   def initialize
 7|     @app = FXApp.new
 8|     @main = FXMainWindow.new(@app, "Hei verden!")
 9|
10|     @hadet_knapp =
11|       FXButton.new(@main, "&Ha det bra!",
12|                     nil, @app,
13|                     FXApp::ID_QUIT )
14|
15|     @hei_knapp = FXButton.new(@main, "Si hei!")
16|     @hei_knapp.connect(SEL_COMMAND) do
17|       puts "Hei verden!"
18|     end
19|   end
20|
21|   def start
22|     @app.create
23|     @main.show  # Vis vinduet.
24|     @app.run    # Start GUI.
25|   end
26|
27| end
28|
29| Verden.new.start
```

[1] http://www.fxruby.org/
[2] http://wxruby.rubyforge.org/
[3] http://ruby-gnome2.sourceforge.jp/
[4] http://ruby-fltk.sourceforge.net/
[5] http://www.fxruby.org/

3.1.1 Innput

[code/fxinnput1.rb]

```
 1| require 'fox'
 2|
 3| class Kalkulator
 4|   include Fox
 5|   def initialize
 6|     @app = FXApp.new
 7|     @main = FXMainWindow.new( @app, "Kalkulator")
 8|     @felt_x = lag_innputfelt("X")
 9|     @felt_y = lag_innputfelt("Y")
10|     @utput  = lag_innputfelt("=")
11|     lag_knapperad
12|     FXButton.new(@main, "Avslutt", nil, @app, FXApp::ID_QUIT )
13|   end
14|   def lag_innputfelt( merkelapp )
15|     ramme = FXHorizontalFrame.new( @main )
16|     FXLabel.new( ramme, merkelapp )
17|     FXTextField.new( ramme, 10, nil, 0, TEXTFIELD_NORMAL | JUSTIFY_RIGHT )
18|   end
19|   private :lag_innputfelt
20|   def lag_knapperad
21|     ramme = FXHorizontalFrame.new( @main )
22|     %w[ + - * / ** ].each do |tegn|
23|       knapp = FXButton.new( ramme, tegn )
24|       knapp.connect(SEL_COMMAND) do
25|         beregn( tegn.intern )
26|       end
27|     end
28|   end
29|   private :lag_knapperad
30|   def beregn( symbol )
31|     x = @felt_x.text.to_f
32|     y = @felt_y.text.to_f
33|     resultat = x.send( symbol, y )
34|     @utput.text = resultat.to_s
35|   end
36|   def start
37|     @app.create
38|     @main.show  # Vis vinduet.
39|     @app.run    # Start GUI.
40|   end
41| end
42|
43| Kalkulator.new.start
```

4 Integrering med C/C++

For å koble Ruby-kode med C/C++ kode kan du: skrive egne utvidelser, innebake Ruby-fortolkeren i programmet ditt, bruke *SWIG*[6] til å pakke inn C/C++ koden, arbeide mot dynamiske bibliotek med DL eller gjøre småtriks med *RubyInline*[7].

4.1 Ruby i C

Ruby er skrevet i C, så man kan enkelt skrive egne metoder i C for å øke hastigheten.

[code/ext_fact1.c]

```
 1| #include "ruby.h"
 2|
 3| VALUE factorial( VALUE self, VALUE index )
 4| {
 5|   VALUE fact = Qnil;
 6|   long i = NUM2LONG( index );
 7|   long f = 1;
 8|   while(i>0){
 9|     f = f * i--;
10|   }
11|   fact = LONG2NUM( f );
12|   return( fact );
13| };
14|
15| Init_ext_fact1()
16| {
17|   rb_define_global_function("factorial", factorial, 1 );
18| };
```

Når utvidelsen er kompilert kan vi hente inn modulen og kalle koden.

[code/ext_fact1.rb]

```
 1| # kompilerer C-koden med:
 2| #   gcc -shared ext_fact1.c -o ext_fact1.so
 3| require 'ext_fact1.so'
 4| puts (1..10).collect{|i| factorial i }.join(', ')
```

[6] http://www.swig.org/
[7] http://www.zenspider.com/ZSS/Products/RubyInline/

4.2 SWIG

SWIG (Simplified Wrapper and Interface Generator) pakker inn C/C++ kode for bruk i andre språk, slik som Ruby, Python, Perl, Java m.m.

Vi finner ut at Vector-klassen fra tidligere ble for treg for våre 3D-beregninger, så vi prøver å lage en C++-variant. Vi har følgende headerfil:

[code/swig_vector.hpp]

```
 1| class Vector {
 2| public:
 3|     double x, y, z;
 4|     Vector( double x = 0, double y = 0, double z = 0 );
 5|     Vector operator+( const Vector & );
 6|     Vector operator-( const Vector & );
 7|     Vector operator*( const Vector & );
 8| };
```

Når vi så skal bruke denne i Ruby, kan vi bruke SWIG til å pakke inn C++-metodene for oss. Vi definerer da en interfacefil:

[code/swig_vector.i]

```
 1| %module graphics
 2| %{
 3| #include "swig_vector.hpp"
 4| %}
 5| class Vector {
 6| public:
 7| %immutable;
 8|     double x, y, z;
 9| %mutable;
10|     Vector( double x = 0, double y = 0, double z = 0 );
11|     Vector operator+( const Vector & );
12|     Vector operator-( const Vector & );
13|     Vector operator*( const Vector & );
14| };
15|
16| %extend Vector {
17|     VALUE to_a(){
18|         VALUE arr = rb_ary_new();
19|         rb_ary_store( arr, 0, rb_float_new(self->x));
20|         rb_ary_store( arr, 1, rb_float_new(self->y));
21|         rb_ary_store( arr, 2, rb_float_new(self->z));
22|         return( arr );
23|     }
24| };
```

Denne kjøres gjennom SWIG og lager en C++-fil med wrapper-funksjoner. Alt kompileres og lenkes:

```
[ code/compile_swig_vector.rb ]
 1| system('g++ -c swig_vector.cpp')
 2| system('swig -c++ -ruby swig_vector.i')
 3| system('g++ -c swig_vector_wrap.cxx -I/usr/lib/ruby/1.8/i586-linux-gnu/')
 4| system('g++ -shared swig_vector_wrap.o swig_vector.o -o graphics.so')
 5|
 6| require 'graphics.so'
 7| p Graphics
 8| p Graphics::Vector
 9| v = Graphics::Vector.new(1,3,7)
10| p v.x, v.y, v.z
```

Nå for den store testen... Kan vi bruke enhetstestene fra Ruby-versjonen av Vector-klassen?

```
[ code/test_swig_vector.rb ]
 1| require 'graphics'    #<- Endret
 2| require 'test/unit'
 3|
 4| class TestVector < Test::Unit::TestCase
 5|   include Graphics    #<- Lagt til
 6| # --- kuttet her ---
```

Jepp, kun to små endringer nødvendig.

4.3 Ruby/DL

DL er et standardbibliotekt gir oss tilgang til den dynamiske lenkeren. Dermed kan vi hente inn og bruke vilkårlige dynamiske biblioteker (DLL og SO) fra Ruby.

```
[ code/dl_libc.rb ]
 1| require 'dl/import'
 2| module Clib
 3|   extend DL::Importable
 4|   dlload "/lib/libc.so.6"
 5|   extern "int printf(char*,int)"
 6| end
 7| Clib.printf("Testing. Tall: %i !\n", 5)
```

4.3.1 Win32API

Win32API er et deprecated bibliotek som gav tilgang til API-funksjoner under Windows. Det er deprecated til fordel for en modul i DL som tilbyr samme funksjonalitet.

```
[ code/dl_win.rb ]
 1| require 'dl/win32'
 2|
 3| message_box = Win32API.new("user32",'MessageBoxA', 'ISSI', 'I')
 4|
 5| parent  = 0
 6| message = "Ok?"
 7| title   = "Error!"
 8| style   = 1 # MB_OKCANCEL
 9|
10| retval = message_box.call( parent, message, title, style )
11|
12| knapp = case retval
13|         when 1 then "OK"
14|         when 2 then "Cancel"
15|         else "(?)"
16|         end
17| puts "Returverdi: #{retval}. Knapp: #{knapp}"
```

4.4 Win32OLE

Har man først nevnt Win32API, bør man få med seg Win32OLE. Det er en mer høynivå innpakking av OLE i Windows.

```
[ code/ole1.rb ]
 1| require 'win32ole'
 2| ie = WIN32OLE.new('InternetExplorer.Application')
 3| ie.visible = TRUE
 4| ie.navigate( "http://www.ruby.no/" )
 5| print "Trykk Enter."
 6| gets
 7| ie.Quit()
```

5 XML

5.1 REXML

REXML[8] er et kraftig verktøy for manipulering av XML som er en del av standardbiblioteket til Ruby.

```
[ code/xml_parsing1.rb ]
 1| require 'rexml/document'
 2| include REXML
 3|
 4| xmltekst = DATA   # Henter data etter __END__
 5|
 6| def rekursiv_print( element, indent = 0 )
 7|   print '  '*indent
 8|   case element
 9|   when Text
10|     print element.to_s.gsub("\n", " ").strip, "\n"
11|   when Element
12|     print element.name, ' - '
13|     element.attributes.each{|key, val|  print key, '="', val, '" ' }
14|     print "\n"
15|     element.each_child do |c|
16|       rekursiv_print(c, indent+1)
17|     end
18|   end
19| end
20|
21| dok = Document.new( xmltekst )
22| rekursiv_print( dok.root )
23|
24| dok.elements.each("/html/body/code"){|elem|  # xpath oppslag
25|   puts elem.children.select{|child| child.kind_of? Text }
26| }
27|
28| __END__
29| <html><title>Ruby-kurs</title>
30| <body>
31| <h1>Introduksjon</h1><p>Bla bla...</p>
32| <h2>Hei verden</h2>
33| <code lang="ruby">puts 'Hei verden'</code>
34| </body></html>
```

[8] http://www.germane-software.com/software/rexml/

Det er også kjekt når man vil konstruere XML-dokumenter.

```
[ code/xml_generering1.rb ]
 1| require 'rexml/document'
 2|
 3| include REXML
 4|
 5| body = Element.new('body' )
 6| body << Element.new('h1') << Text.new('Introduksjon')
 7| body << Text.new("\n")
 8| body << Element.new('p') << Text.new('Hei & ha det!!')
 9|
10| html = Element.new('html')
11| html.add body
12|
13| dok = Document.new
14| dok.add XMLDecl.new( "1.0", "iso-8859-1" )
15| dok.add html
16|
17| puts dok
```

6 Design og arkitektur

6.1 Navnerom og moduler

Moduler kan fungere som navnerom (eng: *namespace*) i tillegg til multippel arv av funksjonalitet.

```
[ code/navnerom1.rb ]
 1| # Moduler som navnerom
 2| module Text
 3|   class Parser ; end
 4|   module SGML
 5|     class Parser ; end
 6|     module HTML
 7|       class Parser ; end
 8|     end
 9|     module XML
10|       class Parser ; end
11|     end
12|   end
13| end
14|
15| # Bruke skop-operatoren
16| htmlparser = Text::SGML::HTML::Parser.new
17|
```

```
18| # Åpne spesifikke navnerom
19| class Text::SGML::HTML::Parser
20|   def non_breakable_space
21|     " "
22|   end
23| end
24|
25| # Hente inn navnerom
26| include Text::SGML
27| xmlparser = XML::Parser.new
```

Du kan også bruke klasser som navnerom i de fleste tilfeller, da `(Class.ancestors.include? Module) == true`, men du kan ikke `include` en klasse, da Ruby ikke har multippel arv fra klasser.

6.2 Singleton

Singleton-metoder eksisterer kun på et objekt. Dette er gjerne klassemetoder, slik som `new`.

```
[ code/singleton_metoder1.rb ]
 1| class EnKlasse
 2|   def self.metode   # Klassemetoder er singleton metoder.
 3|   end
 4| end
 5|
 6| arr = ['usr', 'local', 'bin']     # Nesten alle objekter kan få
 7| def arr.to_str                    # kan få singleton metoder
 8|   sep = File::SEPARATOR           # lagt til seg når som helst.
 9|   File::SEPARATOR + self.join(sep)
10| end
11| puts arr.to_str
12|
13| class <<arr              # Du kan også åpne et objekts
14|   def write( *arg )      # singleton-klasse for å definere
15|     self.push( *arg )    # metoder.
16|   end
17| end
18| arr.write "irb"
19| puts arr.to_str
```

Går vi dypere i metahierarkiet til Ruby, finne vi at singleton-metodene er definert i singleton-klasser.

```
[ code/singleton_klasser1.rb ]
 1| class MinKlasse
 2|   def self.en_metode   # Klassemetoder kan også defineres...
 3|     puts "en"
 4|   end
```

```
 5|   class <<self         # ... i klassens singleton klasse.
 6|     def annen_metode
 7|       puts "annen"
 8|     end
 9|   end
10| end
11|
12| MinKlasse.annen_metode #=> "annen"
```

Singleton-klasser er et "usynlig" mellomledd mellom et objekt og dets klasse, hvor singleton-metodene havner.

(Forventet du noe om et design pattern kalt Singleton her? Titt på modulen `singleton` i standardbiblioteket.)

6.3 Duck typing

"If it walks like duck..."

Ruby benytter seg av en klar distinksjon mellom klasse og type. Duck typing vil si at det meldingene et objekt reagerer på som angir typen, og ikke klassen.

```
[ code/duck_typing1.rb ]
 1| def list_filer( sti )
 2|   Dir.open( sti ) do |dir|       # Dir#open forventer en String,
 3|     puts dir.entries.join(' ')   # men benytter duck typing.
 4|   end
 5| end
 6|
 7| class KatalogSti < Array
 8| end
 9|
10| sti = KatalogSti.new
11| sti << "usr" << "local" << "bin"
12|
13| begin
14|   list_filer( sti )             # Gir konverteringsfeil:
15| rescue TypeError=>err
16|   puts err
17| end
18|
19| class KatalogSti
20|   def to_str                    # to_str er en metode som angir
21|     sep = File::SEPARATOR       # at et objekt kan benyttes som
22|     sep + self.join(sep)        # en String.
23|   end
24| end
25|
26| list_filer( sti )               # Da kan KatalogSti brukes...
```

6.4 Moduler før klasser

Vi kan bare arve fra en klasse om gangen. Dermed er ofte moduler et bedre valg for å samle funksjonalitet som deles. Dette gjelder spesielt dersom den delte funksjonaliteten **konseptuelt** ikke utgjør en superklasse.

```
[ code/modul_arv1.rb ]
 1| module NyttigDill
 2|   def foo; "foo"; end
 3| end
 4| module NyttigDall
 5|   def foo; "bar"; end
 6| end
 7|
 8| class DillDall
 9|   include NyttigDill   # Multippel arv av
10|   include NyttigDall   # funksjonalitet.
11| end
12| puts DillDall.new.foo  #=> "bar"
13|
14| class DillDall
15|   extend NyttigDill    # Utvider klasseobjektet.
16| end
17| puts DillDall.foo      #=> "foo"
```

6.5 Spesiell flytkontroll

6.5.1 throw & catch

For de ganger hvor man ønsker å nøste opp kallstakken og det ikke er et unntak, kan man bruke throw/catch.

```
[ code/throw_catch.rb ]
 1| def som_kaster( level = 0 )
 2|   if level > 0
 3|     som_kaster( level - 1 )
 4|   else
 5|     throw :bunnen_truffet, level
 6|   end
 7|   puts level # blir aldri kalt
 8| end
 9|
10| level = catch(:bunnen_truffet) do
11|   som_kaster( 10 )
12| end
13| puts level
```

6.5.2 Continuation og callcc

Continuations omtales ofte som **kontrollert goto**, men er likevel et vanskelig konsept å vri hjernen rundt.

```
[ code/callcc1.rb ]
 1| def callcc_eksempel
 2|   callcc{|cont|
 3|     puts "Returnerer Continuation"
 4|     return cont
 5|   }
 6|   puts "Avslutter metoden."
 7|   nil
 8| end
 9|
10| cont = callcc_eksempel
11| puts "Ferdig?."
12| cont.call if cont
```

Et mer komplisert eksempel fra Dave Thomas (med kodemassasje gjort at Avi Bryant)

```
[ code/callcc2.rb ]
 1| def with_context(params)
 2|   k, name = catch(:context) {yield; return}
 3|   k.call(params[name] || find_in_context(name))
 4| end
 5| def find_in_context(name)
 6|   callcc{|k| throw(:context, [k, name])}
 7| end
 8|
 9| def update_widget
10|   name = find_in_context(:name)
11|   color = find_in_context(:color)
12|   puts "<#{color}>#{name}</#{color}>"
13| end
14|
15| with_context(   :name => 'dave', :color => 'red' ) do
16|   with_context( :name => 'avi',  :color => 'green' ) do
17|     update_widget               #=> "<green>avi</green>"
18|   end
19|   update_widget                 #=> "<red>dave</red>"
20| end
```

7 Eksempel

7.1 Administrasjon

Et lite skript som lager brukere og hjemmekataloger på en UNIX-maskin.

[code/adduser.rb]

```
 1| #!/usr/bin/ruby
 2| require 'getoptlong'
 3| require 'fileutils'
 4|
 5| class User
 6|   include FileUtils    # henter inn metodene cd, mkdir, chmod m.m i klassen vår.
 7|
 8|   GROUP = "deltaker"               # Vi definerer konstanter for gruppe,
 9|   HOME_BASE = "/home/deltaker"     # katalogplassering og
10|   MODE = 0755                      # filrettigheter.
11|
12|   def initialize( username )
13|     @name = username
14|     raise ArgumentError, "Need username." unless @name   # Må ha et brukernavn.
15|     @homedir =    HOME_BASE + File::SEPARATOR + @name    # Subkataloger...
16|     @public_html = @homedir + File::SEPARATOR + "public_html"
17|     @cgi_bin = @public_html + File::SEPARATOR + "cgi-bin"
18|   end
19|
20|   def make( adduser = false )
21|     make_user if adduser
22|     make_dir( @homedir,     MODE )    # Lager de forskjellige katalogene.
23|     make_dir( @public_html, MODE )
24|     make_dir( @cgi_bin,     MODE )
25|   end
26|
27|   def make_dir( dir, mode = 0755 )
28|     mkdir( dir, :mode => mode ) rescue Errno::EEXIST   # Forsøk å lage katalog.
29|     chmod( mode, dir )             # Sett rettighetene i tilfelle mkdir feilet.
30|     cd( dir )                      # Forflytter oss til katalogen og ...
31|     system("chown #{@name} . * -Rf")         # kjører eksterne kommandoer
32|     system("chgrp #{GROUP} . * -Rf")         # for å endre fileierskap.
33|   end
34|
35|   def make_user                # Kjører eksternt program 'useradd'.
36|     `useradd -g #{GROUP} -d #{@homedir} #{@name} `
37|   end
38|   private :make_user, :make_dir
39|
40| end
```

```
41|
42| if __FILE__ == $0 then
43|   opsjoner = GetoptLong.new(      # Henter kommandolinjeopsjonene.
44|                         ["--username", "-u", GetoptLong::REQUIRED_ARGUMENT],
45|                         ["--adduser",  "-a", GetoptLong::NO_ARGUMENT]
46|                         )
47|   username = nil
48|   adduser = false
49|   opsjoner.each do |opsjon, argument|
50|     case opsjon
51|     when "--username"
52|       username = argument
53|     when "--adduser"
54|       adduser = true
55|     end
56|   end
57|   user = User.new( username )
58|   user.make( adduser )
59| end
```

Merk at vi bruker både `system` og ´ (*backtick*) for å kjøre eksterne kommandoer og programmer.

Et annet alternativ er `IO.popen` som gir deg mer kontroll over innput til og utput fra det eksterne programmet.