

TDP003 Projekt: Egna datormiljön

Installationsmanual för Portföljsystem

IP1 2020

Innehåll

1	Revisionshistorik	1
2	Inledning	2
3	Emacs	2
4	Python3	3
4.1	PIP	3
5	Upprättandet av virtuell miljö	3
6	Flask	5
6.1	Användningsområde	5
6.2	Installation och kontroll	5
6.3	Första Flask-applikationen	5
6.4	Hantering av Flask-applikation	6
6.5	Versionsinformation	6
7	Jinja2	7
8	Git	7
8.1	Konfigurera git	7
8.2	SSH-key till Git	8
8.3	Skapa repositories	8
9	Lägg till och ändra existerande projekt	9
9.1	JSON	9
9.2	Unicode	9
10	GIMP	10
11	Vanliga fel	11
11.1	Fel som kan uppstå i Terminalen	11
11.1.1	GIMP	11
11.2	Fel som kan uppstå vid installation av Emacs	11
11.3	Fel som kan uppstå i Flask	11
11.4	Fel som kan uppstå i Git	12

1 Revisionshistorik

Ver.	Revisionsbeskrivning	Datum
1.1	Första utkastet av installationsmanualen	20-09-22
1.2	Innehållsförteckning, rubriker och underrubriker tillagda	20-09-23
1.3	Första versionen av inledningen skriven	20-09-23
1.4	Första versionen av virtuell miljö skriven	20-09-23
1.5	Första versionen av Python3 installationsguiden skriven	20-09-23
1.6	Python3 installationsguide uppdaterad	20-09-23
1.7	Förklaring av den virtuella miljön uppdaterad	20-09-23
1.8	Inledning uppdaterad	20-09-23
1.9	Första versionen på installation av Emacs skriven	20-09-23
1.10	Förklaring på installation av Emacs uppdaterad	20-09-23
1.11	Inledning uppdaterad	20-09-23
1.12	Förklaring på installation av Emacs uppdaterad	20-09-23
1.13	Första versionen på installation av GIMP skriven	20-09-23
1.14	Första versionen på installation av Flask skriven	20-09-23
1.15	Första versionen på installation av Git skriven	20-09-23
1.16	Fel som kan uppstå i terminalen skriven	20-09-23
1.17	Förklaring av den virtuella miljön uppdaterad	20-09-23
1.18	Förklaring på installation av Flask uppdaterad	20-09-23
1.19	Förklaring av unicode och projekthantering skriven	20-09-23
1.20	Förklaring av JSON skriven	20-09-23
1.21	Vanliga fel i Flask uppdaterad	20-09-23
1.22	Vanliga fel i Git uppdaterad	20-09-23
1.23	Installation av Jinja2 skriven	20-09-23
1.24	Första versionen av revisionshistoriken skriven	20-09-24
1.25	Terminalkommandon omformaterade	20-09-24
1.26	Text om SSH-key till git skriven	20-09-24
1.27	Bildredigering gjord	20-09-24
1.28	Korrigerig av meningsbyggnad	20-09-24
1.29	Fel som kan uppstå i Git skrivet	20-09-24
1.30	Terminal kommandon modifierade	20-09-24
1.31	Uppdatering av installation av python3	20-09-29
1.33	Listan i inledningen fixad	20-09-29
1.34	Andra versionen av installation av Flask skriven	20-09-29
1.35	Komplettering till delen om virtuell miljö	20-09-29
1.36	Uppdateringskommando redigerat	20-09-29
1.37	Redigering av titeln	20-09-29
1.38	Git kommandon, SSH-key och disposition fixad	20-09-30
1.39	Namngivning fixad	20-09-30
1.40	Rubrik för versionsinformation lades till	20-09-30
1.41	Redigering av JSON- och unicode-bild	20-10-01
1.42	Ändring av kommando formatering	20-10-01
1.43	Onödiga radbrytningar togs bort	20-10-01
1.44	Referenser i texten fixades	20-10-01
1.45	Andra versionen av revisionshistoriken skriven	20-10-01

2 Inledning

Denna installationsmanual beskriver hur man installerar de program och paket som krävs för att skapa och underhålla sin egen projektportfölj. Författare av detta dokument vill dock understryka att Emacs som editor per se inte krävs, men den rekommenderas på grund av sin stora funktionalitet och då programvaran är open source. Program och paket som behövs är: Emacs, Python3, Jinja2, Flask, Git, och GIMP. Alla installationer kommer att ske via terminalen.

Minimikraven för att datorn ska kunna installera samt köra Linux är:

- 25 GB Diskutrymme
- 700 MHz CPU
- 4 GB RAM

Ett Linux operativsystem är obligatoriskt för att följa denna installationsmanual. Detta dokument utgår ifrån Linux distributionen Ubuntu 20.04.

Innan installation av nya program bör man göra en uppdatering av systemet. Detta för att få den senaste versionen av alla program och minimera installationskonflikter.

1. Skriv kommandot:

```
sudo apt update && sudo apt upgrade
```

Skriv in adminlösenord vid behov.

3 Emacs

Emacs är en textredigerare med öppen källkod som kommer användas för att skriva all kod som krävs för portfolion. Emacs är inte språkbegränsat och har många hjälpsamma funktioner. Du bör bekanta dig med Emacs då programmet inte använder de främst förekommande tangentbords-kombinationerna som till exempel Ubuntu eller Windows använder.

1. För att installera Emacs används följande kommando i ett terminalfönster:

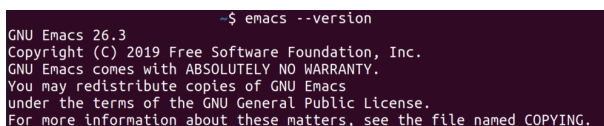
```
sudo apt install emacs
```

Skriv in lösenord vid behov.

2. För att kontrollera att Emacs installerades korrekt skriver du i terminalen:

```
emacs --version
```

Om det ser ut som figur 1 har du lyckats med installationen:



```
-$ emacs --version
GNU Emacs 26.3
Copyright (C) 2019 Free Software Foundation, Inc.
GNU Emacs comes with ABSOLUTELY NO WARRANTY.
You may redistribute copies of GNU Emacs
under the terms of the GNU General Public License.
For more information about these matters, see the file named COPYING.
```

Figur 1:

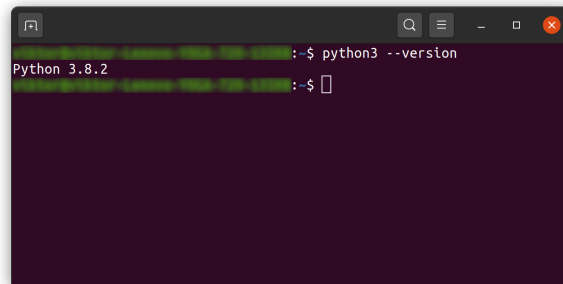
Emacs kommer i två versioner: en GUI(Graphical User Interface) baserad version där du kan klicka dig fram med musen och har ikoner i toppen för att utföra vissa funktioner, samt en "terminal-liknande" version som endast fungerar med hjälp av kommandon. Den installationen du slutfört har installerat båda versionerna.

Har du stött på problem? Vanliga fel vid installation av Emacs nämns i rubrik 10.2.

4 Python3

1. Python3 kommer förinstallerat till Ubuntu, kontrollera detta i terminalen med kommandot:

```
python3 --version
```



Figur 2:

2. Skulle Python3 inte vara installerat, görs detta med kommandot:

```
sudo apt install python3
```

Kontrollera sedan om det installerats:

```
python3 --version
```

4.1 PIP

PIP är en paketinstallerare till Python som används till att installera Pythonpaket såsom Flask och Jinja2. Installera PIP med kommandot:

1. Installera PIP med kommandot:

```
sudo apt install python3-pip
```

2. PIP kan sedan uppdatera sig själv med kommandot:

```
pip install --upgrade pip
```

5 Upprättandet av virtuell miljö

Pythons standardförfarande när det kommer till tredjepartsutvecklade paket är att de alla installeras under samma directory. Man kan kolla vart de är installerade genom att göra följande:

1. Starta en terminal och skriv:

```
python3
```

2. Importera *site* genom att skriva:

```
import site
```

3. För att hämta sökvägar, skriv:

```
site.getsitepackages()
```

Då paketen sparats i dist-packages directoryn kan inte Python särskilja dem som olika versioner utan bara per namn. Detta blir intressant när man börjar hantera ett flertal olika projekt som då förmodligen kommer vara beroende av olika versioner av en eller flera tredjepartsutvecklade paket vilket då inte är möjligt för alla Python projekt måste använda det som finns i dist-packages.

Detta problem löses genom att använda virtual environments som fungerar som separata Pythoninstallationer som då får sina egna dist-packages directories. Man kan ha ett godtyckligt antal virtuella miljöer och bör således användas för varje projekt för att minimera problemen med beroenden mellan projekt.

Venv som står för "virtual environments" är det sättet vi implementerar virtuella miljöer för våra projekt på i Python och nedan följer en instruktion på hur du upprättar ett directory med en virtuell miljö genom att använda *venv*.

1. Python3 bör ha *venv* installerat från början i sitt standardbibliotek men om du inte har det ladda ned *venv* genom att ange:

```
sudo apt-get install python3-venv
```

Ange lösenord för att styrka att du är sudo användare och tryck:

```
y
```

2. Skapa ett directory med kommandot:

```
mkdir Portfolio
```

3. Flytta dig till ditt directory med kommandot:

```
cd Portfolio
```

4. Skriv i terminalen:

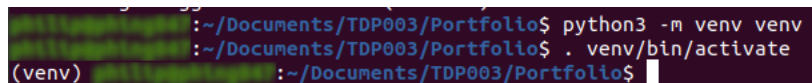
```
python3 -m venv
```

för att ange var ditt directory för den virtuella miljön ska vara där den virtuella Pythoninstallationen kommer finnas.

5. Skriv:

```
. venv/bin/activate
```

för att aktivera den virtuella miljön. Det går att aktivera den vart som helst. Nu kommer det stå "(venv)" i terminalen framför ditt namn och du är inne i den virtuella miljön.



```
~/Documents/TDP003/Portfolio$ python3 -m venv venv
~/Documents/TDP003/Portfolio$ . venv/bin/activate
(venv) ~/Documents/TDP003/Portfolio$
```

Figur 3:

När du är i den virtuella miljön, vilket indikeras av *venv* till vänster om ditt namn som ovan, går det att installera paket genom PIP3. Då kommer de paketen endast att finnas inom den virtuella miljön.

För att avsluta *venv*, ange i terminalen:

```
deactivate
```

6 Flask

6.1 Användningsområde

Flask är ett ramverk för att utveckla och hantera webb-baserade applikationer med Python. I detta fall används Flask för att hantera Jinja2 och testa portfolion innan vi publicerar den. Flask hjälper till att få Jinja2 och andra bra bibliotek för webbutveckling att arbeta tillsammans. Flask tillhandahåller även en lokal testserver och debugger.

6.2 Installation och kontroll

1. Börja med att aktivera Venv (Virtual enviroment) enligt tidigare instruktioner. Med terminalen i Venv skriver du:


```
pip3 install Flask
```

för att installera Flask.

2. För att kontrollera om Flask installerats korrekt skriver du:

```
flask --version
```

utan att lämna din Venv. Kontrollera dina versions nummer mot figur nedan. Dyker inte versions nummer upp i terminalen eller om det är fel versions nummer, se felhantering **11.3 Fel som kan uppstå i Flask**.



```
Python 3.8.2  
Flask 1.1.2  
Werkzeug 1.0.1
```

Figur 4:

6.3 Första Flask-applikationen

När Flask har installerats, skapa en enkel Flask-applikation för att testa att Flask fungerar korrekt.

1. Via terminalen (du kan fortfarande ha Venv aktiverat) skapa en projektmapp genom kommandot:

```
mkdir [mappens namn]
```

2. Navigera in i mappen och skriv följande i terminalen. Kommandot både skapar och öppnar filen:

```
emacs hello.py &
```

(& gör att du inte behöver stänga Emacs för att fortsätta använda terminalen).

3. Kopiera följande text till *hello.py*:

```
from flask import Flask  
app = Flask(__name__)  
  
@app.route('/')  
def hello_world():  
    return 'Hello World!'
```

6.4 Hantering av Flask-applikation

Om filen är nersparad, försätter du med följande steg, som görs varje gång en Flask applikation startas:

1. När filen är sparad skriv:

```
export FLASK_APP=hello FLASK_ENV=development
```

i terminalen.

2. Följt av kommandot

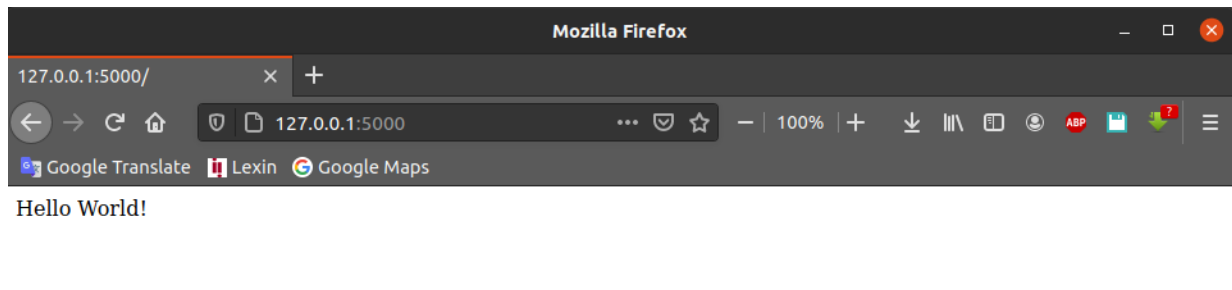
```
flask run
```

för att köra Flask-koden.

3. Slutligen öppna upp en webbläsare och skriv in URL:en

```
http://127.0.0.1:5000
```

för att se Flask-applikationen. Om du använder mozilla firefox bör det se ut som i figuren nedan.



Figur 5:

4. När du vill stanna Flask-applikationen tryck *Ctrl + C* i terminalen med Flask-applikationen igång.
5. Sedan bör Python-Venv avaktiveras med kommandot:

```
deactivate
```

6.5 Versionsinformation

När du jobbar med större projekt eller kanske i grupp med andra personer kan det vara bra att skapa ett dokument som innehåller all versionsinformation om de bibliotek du använder i projektets virtuella miljö. Detta kommer göra att andra användare lätt kan hämta de nödvändiga biblioteken för projektet men även göra så att du i framtiden inte får konflikter som följd av att nya versioner av samma bibliotek fungerar annorlunda tillsammans med projektet. För att skapa en sådant här versionsinformationsdokument gör du följande:

1. PIP har ett verktyg som skriver ut versionsinformation om de bibliotek som används i din utvecklingsmiljö. Vi använder det verktyget och skriver informationen till ett textdokument, 'requirements.txt'. Ställ dig i projektets root-katalog och skriv följande i terminalen:

```
pip3 freeze > requirements.txt
```

2. Du har nu ett textdokument med all versionsinformation om dina bibliotek som du behöver. Vill du på en annan dator eller i en ny miljö arbeta med projektet behöver du bara skriva:


```
pip3 install -r requirements.txt
```

7 Jinja2

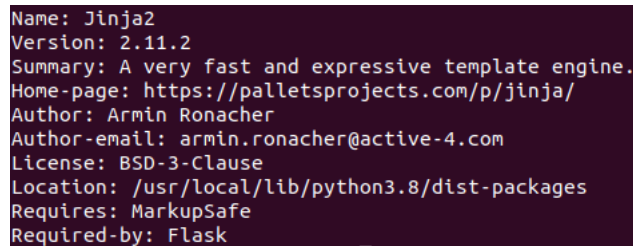
Jinja2 är ett underbibliotek till Flask och används bland annat för att skapa mallar av html-sidor. Man kan till exempel med hjälp av Jinja se till att en navbar och footer automatiskt dyker upp på alla sidor, utan att man på egen hand behöver kopiera över koden. Det kommer att användas för att fylla portfolion med relevant information från datalagret.

Jinja2 bör ha installerats samtidigt som Flask, men för att försäkra sig om det kan man aktivera Venv och sedan göt följande:

1. I terminal skriv:

```
pip3 show jinja2
```

Om det finns kommer följande text att dycka upp i terminalen:



```
Name: Jinja2
Version: 2.11.2
Summary: A very fast and expressive template engine.
Home-page: https://palletsprojects.com/p/jinja/
Author: Armin Ronacher
Author-email: armin.ronacher@active-4.com
License: BSD-3-Clause
Location: /usr/local/lib/python3.8/dist-packages
Requires: MarkupSafe
Required-by: Flask
```

Figur 6:

2. Ifall den texten inte förekommer kan du installera Jinja2 med kommandot:

```
pip3 install jinja2
```

8 Git

Git är ett versionshanteringsprogram där användaren kan spara sin nuvarande version av en fil. Det går även att spåra ändringar, användaren har därför möjlighet att gå tillbaka till en tidigare version av filen som är sparad. På Git kan repository skapas. I repository får användaren en god översikt över sina filer och kan även dela dessa med andra användare. Detta gör det möjligt att arbeta på projekt tillsammans.

8.1 Konfigurera git

1. Kontrollera om Git är installerat i terminalen.

```
git --version
```

2. Om Git inte är installerat skriv följande i terminalen för att installera Git. I detta steg kan användaren behöva skriva sitt lösenord, en bekräftelse av installationen kommer även krävas. Detta sker genom att skriva 'Y' i terminalen när det efterfrågas.

```
sudo apt install git
```

3. Konfigurerar Git med användarnamn och epost.

```
git config --global user.name "Namn Efternamn"
git config --global user.email "email@mail.se"
```

8.2 SSH-key till Git

Git har SSH-nycklar som försäkrar Git serverna att datorn är användarens dator och kommer därför aldrig ifrågasätta uppladdningarna med Gits auktorisering. Om du känner dig trygg med att inte ha alla uppladdningarna lösenordsskyddade eller vill ha ett unikt lösenord för alla uppladdningar från denna dator till ditt Git konto, kan du följa vidare instruktioner:

1. Först ska du via Linux SSH-keygen program generera en SSH-nyckel genom att skriva kommandot:

```
ssh-keygen -t rsa -C "GitLab" -b 4096
```
2. Sedan kommer du få ett alternativ vart du vill spara SSH-nycklarna, välj default genom att trycka ENTER.
3. Om du vill ha ett eget lösenord för uppladdning till Git från denna dator, skriv in lösenordet och tryck på ENTER. Om du inte vill ha lösenordsskyddat var noggrann att enbart trycka ENTER, för att förhindra att ett okänt lösenord råkas skrivas in.

Om nyckeln har genereras ska utskriften likna:

```
The key's randomart image is:
+---[RSA 4096]-----+
|  o\%B=. .          |
|  .o 0+.o          |
|  ..E=...          |
|  .oo.+            |
|  .=.S .           |
|  ..++ .           |
|  . ...o...        |
| .o ..  +o.o       |
```

4. Sedan ska du kopiera nyckeln (all text) i filen *id_rsa.pub* som ligger i den dolda mappen */.ssh/*. (Tips: *Ctrl + H* i filhanteraren *Files* visar dolda mappar i Linux)
5. Lägg till nyckeln online på Git SSH-keys sida. SSH-keys sidan borde vara tillgänglig i inställningar för användaren. När detta är klart har du satt upp ett lösenord eller tagit bort lösenordsskyddet för alla uppladdningar till ditt Git konto. Från och med nu ska alla Git uppladdningar till din angivna användare kräva lösenordet du angav eller inte fråga om något lösenord.
6. Om du vill ha samma lösenord till ett annat Git konto behöver du bara lägga till samma nyckel i det kontots SSH-keys på Git hemsidan. OBS! Det går inte att knyta samma nyckel till olika Githemsidor!

Mycket viktigt! Filen *id_rsa* (inte *id_rsa.pub*) får ingen utom du själv ha eftersom det ger vem som helst koppling till din dator. Att utelämnas denna SSH-key är desamma som att fysiskt ge bort datorn med lösenord till en okänd!

8.3 Skapa repositories

1. Användaren behöver ställa sig i en lokal katalog, där kommer filerna från Git att hamna. Kommandot nedan ska sedan köras.

```
git init
```

2. Användaren har nu skapat en synkronisering, lägg nu till repository där filer kommer hämtas ifrån. Det kan se ut som nedan exempelvis.

```
git remote add origin git@gitlab.liu.se:<liuid>/'lokal katalog.git'
```

3. För att hämta ett repository kan man använda git clone”kommandot.

```
git clone git@gitlab.liu.se:<liuid>/<namn>.git
```

Det hämtade reposetorit bör nu finnas i din lokala respository eller mapp.

9 Lägg till och ändra existerande projekt

Nya projekt sparas i en JSON-fil med UTF-8 teckenkodning. Ändringar i projekt görs också manuellt i samma JSON-fil. Förändringarna uppdateras automatiskt till webbsidorna utan att det behövs omstart av webbserver.

9.1 JSON

JSON står för **J**ava**S**cript **O**bject **N**otation och det används ofta när data skickas från en server till en webbsida. De största fördelarna med JSON är att den är kompakt och kan även läsas av människor. JSON kan användas i kombination med många programmeringsspråk, det är inte som namnet antyder exklusivt för java. JSON modulen i Python3 har som standard UTF-8 som teckenkodning. JSON filerna måste ha denna teckenkodning för att systemet ska fungera.

1. Här är några grundläggande JSON-syntaxregler:

- Data lagras i namn/värdepar.
- Data separeras med komma.
- Måsvingar lagrar objekt.
- Hakparanteser lagrar matriser(arrays).

Figur 7 är ett exempel på hur ett JSON objekt ser ut.

```
{ "employees": [  
  { "name": "Shyam", "email": "shyamjaiswal@gmail.com" },  
  { "name": "Bob", "email": "bob32@gmail.com" },  
  { "name": "Jai", "email": "jai87@gmail.com" }  
]
```

Figur 7: JSON objekt exempel

9.2 Unicode

I början av internet fanns bara ASCII teckenkodning. Detta var eftersom allt som skulle behövas var kontrolltecken, skiljetecken och siffror. Nu i en tid av sociala medier och global interkommunikation räcker det inte. Detta eftersom det nu förekommer tecken från olika alfabet på en och samma webbsida, det behövdes en annan standard. Det är därför unicode skapades. Unicode hjälper datorer att förstå hur de borde hantera

text skriven i olikaskriftssystem. Unicode består av ett repertoar med fler än 100 000 olika skrivtecken som har en unik teckenkod. UTF-8 är det formatet av unicode som är standard för representation av text på webbsidor. Som referens är figur 8 en ascii-tabell av alfabetet som enbart är en liten del av UTF-8:s utbud.

Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

Figur 8: Alfabetets teckenkodning enligt ascii-formatet

10 GIMP

För att redigera bilder så de får lämpligare storlek och form kan en bildredigerare användas. GIMP är ett kostnadsfritt alternativ med öppen källkod.

1. För att installera GIMP skriv kommandot:

```
sudo snap install gimp
```

Kommandot installerar GIMP med hjälp av pakethanteraren *snap*.

2. För kontrollera att du installerat programmet, skriv:

```
gimp --version
```

Detta skriver ut vilken version som finns installerad.

11 Vanliga fel

11.1 Fel som kan uppstå i Terminalen

När du använder *Pip* och *Python* i terminalen utan *Venv* aktiverat kommer du behöva använda kommandona `pip3` och `python3`. När *Venv* är aktiverat behöver du istället skriva `pip` och `python`.

11.1.1 GIMP

Kontrollera att du har en stabil internetuppkoppling när du installerar program.

1. För att säkerställa att du har en uppdaterad version av tillgängliga paket skriv kommandot:

```
sudo apt update
```

2. Om programmet inte fungerar som förväntat kan det vara värt att avinstallera och installera om. För att avinstallera `snap` skriv kommandot:

```
sudo snap remove gimp
```

Om problem uppstår under installationen vilka inte är relaterade till de programspecifika felen (se senare rubriker) kan det finnas olika anledningar.

- Se till att rätt kommando matats in i terminalen. Stavning felkopiering kan orsaka att kommandon inte körs korrekt.
- Kontrollera om internetuppkoppling är etablerad. Saknas internetuppkoppling kan inget installeras från webben. Vänligen kontrollera internet anslutning.
- Se till att all installerad programvara är uppdaterad. För att uppdatera program installerade på datorn kör kommandot `sudo apt update` i terminalen.

11.2 Fel som kan uppstå vid installation av Emacs

Vissa får ett problem med att "`sudo apt-get install emacs`" inte fungerar. Då kan man istället använda sig av ett annat kommando för att installera Emacs:

1. Skriv i terminalen:

```
snap install emacs
```

Efter detta kommer Emacs vara installerat på din dator.

11.3 Fel som kan uppstå i Flask

- Ifall Flask inte skulle fungera kan det bero på att internetuppkoppling saknas eller att `sudo apt update` inte körs innan installationen (vilket håller Flask uppdaterad).
- *Venv* ska användas under installationen av Flask. Ifall `pip3` kommandot istället används är det ingen garanti att Flask kommer installeras eller att Flask sedan kommer fungera.
- Skulle URL `http://127.0.0.1:5000` inte fungera kolla vilken URL som terminalen har angivit efter start av Flask. Det fungerar även att gå in på `http://localhost:5000` istället, vilket minimerar risken att fel siffror skrivs in.
- Filen måste heta `hello.py` för att den angivna koden ska fungera. Det finns inga vägar runt det om du vill använda exempelkoden!
- Om servern inte startar med flask run kan du ha glömt att skriva `export FLASK_APP=<FILNAMN>.py`.

- Skulle `FLASK_ENV=development` hoppas över vid start av Flask kommer applikationen att krascha när servern körs på grund av att Flask kräver utvecklingsmiljö.
- Skulle det fortfarande inte fungera kontrollera då att ingen annan server körs på port 5000 på din dator samtidigt.

11.4 Fel som kan uppstå i Git

- Ett felmeddelande som nämner “Permission denied” vid användning av `git clone` eller `git remote add` beror troligen på att felaktig SSH-nyckel har getts till GitLab. Kontrollera att SSH-nyckeln `id_rsa.pub` har lagts in i sin helhet i GitLabs SSH-inställningar. Kontrollera även att SSH-nyckeln som används har skapats på nuvarande dator och operativsystem.
- För att undvika vissa fel i repos med flera medlemmar, som ett felmeddelande vid försök att pusha till master-grenen, bör rollen “maintainer” eller högre ges till användarkontot.
- Felmeddelandet “not a git repository” beror på att felaktig URL har skrivits in i kommandot. För att undvika fel bör URL:en kopieras från “Clone with SSH” under knappen Clone i repots startsida.
- Vid användning av `git push` eller `git pull` utan att specificera en plats kan ett felmeddelande som säger “current branch master has no upstream branch” dyka upp. För att lösa detta och undvika att behöva specificera plats vid varje push eller pull bör kommandot `git branch --set-upstream-to=origin/<branch> master` användas, där `<branch>` är den grenen som ska pushas till (t.ex master för huvudgrenen i ett projekt).