Practical 1
Introduction to Linux and Python

Learning Objectives

1. Define and use basic datatypes and control structures in Python
2. Define and use key commands in the Linux operating system
3. Use Python in a Linux environment to model simple systems

Overview

In this practical you will be using the most common commands in Linux and learning to use a text editor. With these skills, you can write your first few Python programs. In this case we will write a program to implement a simple systems dynamics model.

Tasks

1. Logging on to Linux

Use your Oasis username and password to login to the lab computers. Now open a terminal window, can you work out where the application for a terminal window is?

In this unit you MUST use the terminal for all navigation, and manipulation. This includes using terminal based text editors.

Always lock your machine when you leave it!

Apart from terminal windows, the other main application you will be using is a web browser. On the Linux machines you should look for **Firefox** and run it. From there you can get to Blackboard and other websites.

2. Introduction to Linux

Using only the terminal it is now time to construct a directory structure you can use to keep the semesters work organised. A sample of the Unix commands available to you have been provided below. There are many more commands you will use through out the unit, the < > braces are just a convention to show something that you fill in. Note that in Unix/Linux folders are referred to as directories.

Try typing the following commands one at a time and check what they do…

```
ls
ls –l
pwd
mkdir test
ls
cd test
ls
ls –la
```

Command	Description	Examples
ls	Lists all files in the current directory, if an argument is provided lists all files in the specified directory.	ls ls –l ls –la
cp <src> <dest>	Copies the file from source to destination.	cp Downloads/test.py . cp test.py test2.py
mv <src> <dest>	Moves the file from source to destination. If the destination ends in a file name it will rename the file.	mv test.py Prac1 mv test.py mytest.py
pwd	Lists the directory you are currently in (Print Working Directory)	pwd
cd <dir>	Moves to <dir>	
mkdir <dir>	Creates a new directory	mkdir Prac1
rm <filename>	Removes a file	rm fireballs.py
rmdir <dir>	Removes a directory (must be empty first)	rmdir Prac12

Using these commands create the following directory structure within your home directory. A **How to use Unix** and cheatsheet document has been uploaded to Blackboard, it will be helpful if you get stuck.

- FOP
  - Prac01
  - Prac02
  - Prac03
  - Prac04
  - Prac05
  - Prac06
  - Prac07
  - Prac08
  - Prac09
  - Prac10
  - Prac11

The current directory is referenced by a single fullstop (.), the parent directory is referenced by two fullstops (..) and all pathways are relative to the current location.

3. Introduction to the Text Editor (vim)

We are going to use the **vim** text editor to create your README file for Prac01. Vim is an enhanced version of vi – a visual, interactive editor. There is an **Introduction to Vi** document and a “cheat-sheet” on Blackboard.

Change directory into the Prac01 directory (cd will get you back to your home directory) and create the README file:

```
> cd Prac1
> vim README
```

You will now be in the vim text editor with an empty file. Vim has two modes – command mode (where you can move around the file and use commands) and insert

text mode. Type “i” to go into insert mode and type in the following README information for Practical 1.

```
## Synopsis

Practical 1 of Fundamentals of Programming COMP1005/5005

## Contents

README – readme file for Practical 1

## Dependencies

none

## Version information

<today’s date> - initial version of Practical 1 programs
```

Press <esc> to exit insert mode, then `:wq` to save the file (w) and exit vim (q).

Type `ls -l` and you will see that you have created a file called README, and it has a size and a date. We will make README files for all of our practicals to hold information about the files in that directory.

4. Welcome to Python!

Below is a simple program to get you used to the editor and running python scripts. To create a file for the program, type:

```
vim hello.py
```

Then type in the following code… It is important to type it yourself and not copy/paste – this is how you will learn and remember!

```
#
# hello.py: Print out greetings in various languages
#

print('Hello')
print("G'day")
print('Bula')
print("Kia ora")
```

To run the program, type:

```
python3 hello.py
```

You will probably get an error message as a response (unless you typed it in perfectly). Don’t worry, check through your code for the error and try running it again. Go back into the file and make corrections – use the cursor keys to get to the position (<lineno>G). Some handy editing commands are:

- to delete a character type “`x`”

- to delete a line "`dd`"
- to delete a word "`dw`"
- to change a word "`cw`"
- to insert/append after the end of the current line, type "`A`"
- to undo the last command, type "u"
- to redo the last command, type "."

Save the file and try to run it again. If you're having trouble, ask your tutor, or even the person next to you, to see if they can find what's wrong. Sometimes it takes someone else's fresh and/or experienced eyes to see an error. This is called "debugging" and the reward comes when the code finally runs!

Try adding some more greetings of your own...

5. Let's cut some code!

Below are three programs, each extending on the previous one. To create the first program, type:

```
vim numbers1.py
```

```
#
# numbers1.py: Read in number and echo the number
#              and its type to screen:

print('Enter a number...')
numberstr = input()
print('Number =', numberstr, ' Type : ', str(type(numberstr)))

number = int(numberstr)
print('Number =', number, ' Type : ', str(type(number)))
```

To run the program, type:

```
python3 numbers1.py
```

Notice how the two print statements print the same number (well, it looks the same), but their variable types are different? Everything you read in will be a string. If you want a number, you'll need to convert it with the int() or float() functions.

numbers2.py

Now, make a copy of your first program and call the copy numbers2.py:

```
cp numbers1.py numbers2.py
```

Edit ~~numbers.py~~ numbers2.py to match the code below:

```
#
# numbers2.py: Read in ten numbers and give sum of numbers
#
print('Enter ten numbers...')
total = 0
```

```
for i in range(10):
    print('Enter a number (', i, ')...')
    number = int(input())
    total = total + number

print('Total is ', total)
```

Save and exit the file and try running it. What are the values that variable "i" holds each time through the loop? How would you change the **for loop** in the program to request five numbers be entered.

Have you noticed that the code changes colour as you type? This is syntax highlighting. When you're in command mode in vim you can type `:syntax off` and `:syntax on` to turn it off/on.

numbers3.py

Our final modification of the program will be numbers3.py. Copy numbers2.py to numbers3.py and edit it to match the following:

```
#
# number3.py: Read in a list of numbers (negative to exit) and
#             give the sum of the numbers

count = 0
total = 0
print("Enter a list of numbers, negative to exit...")
number = int(input())

while number >= 0:
    count += 1           #equivalent to count = count + 1
    total += number      #equivalent to total = total + number
    print("Next number...")
    number = int(input())

print("Total is ", total, " and count is ", count)
```

Save and exit and then run numbers3.py. How would you need to change the **while loop** in the code to have it exit on zero instead of negative numbers?

6. Now for a simple systems model… Unconstrained Growth and Decay

From the "Introduction to Computational Science" text:

"Many situations exist where the rate at which an amount is changing is proportional to the amount present. Such might be the case for a population, say of people, deer, or bacteria. When money is compounded continuously, the rate of change of the amount is also proportional to the amount present. For a radioactive element, the amount of radioactivity decays at a rate proportional to the amount present."

So, growth and decay models are common in many domains. We will implement algorithm 2 from Module 2.2 of the text book (p25). Chapter 2 is available for download at http://press.princeton.edu/titles/10291.html, and provides background to these types of models.

We are going to write Python code for simulating unconstrained growth based on the following *pseudocode* from the text:

 # Algorithm 2 - simulation of unconstrained growth

initialise simulation length
initialise population
initialise growth rate
initialise (length of) time step
number of iterations = simulation length / time step
growth rate (per step) = growth rate * time step

for i = 0 to number of iterations-1 do
    growth = growth rate (per step) * population
    population = population + growth
    time = i * time step
    display time, growth, population

Compare this to the following code. We are implementing the scenario which follows the Algorithm on page 25.

```
#
# growth.py - simulation of unconstrained growth
#

print("\nSIMULATION - Unconstrained Growth\n")

length = 10
population = 100
growth_rate = 0.1
time_step = 0.5
num_iter = length / time_step
growth_step = growth_rate * time_step

print("INITIAL VALUES:\n")
print("Simulation Length (hours): ", length)
print("Initial Population: ", population)
print("Growth Rate (per hour): ", growth_rate)
print("Time Step (part hour per step): ", time_step)
print("Num iterations (sim length * time step per hour): ",
num_iter)
print("Growth step (growth rate per time step): ",
growth_step)

print("\nRESULTS:\n")
```

```
print("Time: ", 0, " \tGrowth: ", 0, " \tPopulation: ", 100)

for i in range(1, int(num_iter) + 1 ):
    growth = growth_step * population
    population = population + growth
    time = i * time_step
    print("Time: ", time, " \tGrowth: ", growth,
          " \tPopulation: ", population)

print("\nPROCESSING COMPLETE.\n")
```

Type in the code and run it. Can you see why the for loop was changed from 0 to num_iter to 1 to num_iter+1?

7. Updating the README

You now have five programs in the Prac1 directory. Enter the name of each of them along with a description under “Contents” in the README file.

8. Making a zip file

To bundle up and compress files we can use zip/unzip. Similar programs are tar (Tape Archive) and gzip (GNU zip).

To make a zipped file for Practical 1, go to the Prac1 directory inside your FOP directory. Type pwd to check that you are in the right place.

Create the zip file by typing:

```
zip Prac1_<your_student_ID> *
```

This will create a file `Prac1_<your_student_ID>.zip` which includes everything in your current directory – four programs and the README. Check the contents of the zip file by typing:

```
unzip –l Prac1_<your_student_ID>.zip
```

Submission

All of your work for this week’s practical should be submitted via Blackboard using the link in the Week 1 unit materials. This should be done as a single "zipped" file. This is the file to submit through Blackboard. There are no direct marks for these submissions, but they may be taken into account when finalising your mark.

Go to the Assessment link on Blackboard and click on Practical 1 for the submission page.

Reflection

1. **Knowledge**: What are the two modes in vi / vim?
2. **Comprehension**: What is the name of the lab machine you are working on? Hint: use the hostname command or look at the prompt.
3. **Application**: What series of commands would you need to go to the directory FOP/assignment in your home directory and compress all the files?
4. **Analysis**: What variable would you change in growth.py to have more iterations (steps) per hour?
5. **Synthesis**: How would you code a for loop to print “Hello World!” 15 times?
6. **Evaluation**: What part of the prac did you find most challenging? (You can give feedback to the lecturer/tutor…)

Challenge

For those who want to explore a bit more of the topics covered in this practical.

1. Have a look at other problems from the text book: http://press.princeton.edu/titles/10291.html
2. Work through a Linux tutorial
3. Work through a vi/vim tutorial
4. Read some samples of README files for large projects - https://github.com/matiassingers/awesome-readme