FEEG6002 Revision Notes

January 2020

Part I

Applied Programming

Windows vs UNIX commands

- Subtle differences between the commands

Windows	UNIX	Description
`cd <dir>`	`cd`	change directory
`dir <dir>`	`ls`	list contents of current directory
`help`	`help`	get help
`rename <old> <new>`	`mv <old> <new>`	rename directory
`mkdir <dir>`	`mkdir <dir>`	make directory
`move <from> <to>`	`mv <from> <to>`	move file
`del <dir/file>`	`rm (-rf) <dir>`	delete file/directory
`chdir`	`pwd`	prints working directory
`type`	`cat`	prints file contents
`copy <from> <to>`	`cp <from> <to>`	copies file
`cmd.exe`	`./cmd`	executes program

- Redirecting stdout to a file: `echo 1 2 3 4 > nums.txt`
- Appending stdout to a file: `./printtable >> res.txt`
- redirecting stdin: `./calcquares < nums.txt`
- Redirecting both:

Emacs

- Old but powerful text editor.
- Can be used with GUI or command line.

- Commands:

Command	Description
`C-x C-f`	open or create file
`C-x C-s`	save file
`M-x compile`	type command (e.g. compile)
`M-x eshell`	run code in eshell
`C-x C-c`	quit Emacs

Linux terminal

Understanding how to use a Linux terminal is important skill: it allows you operate Linux devices without a GUI, such as a Raspberry Pi or, at the opposite end of the spectrum, a high performance supercomputer.

- In most OS environments, connecting to a remote machine involves typing

  ```
  ssh -X user@feeg6002.soton.ac.uk
  ```

- There are a number of ways of transferring files to a remote linux server; One of which is using `scp` (secure copy)

  ```
  scp file.c user@feeg6002.soton.ac.uk:/remote/path
  ```

  N.B. The path is optional

- Emails can be sent directly from the cmd line using `mail`

  ```
  echo "body" | mail -s "Subject" email@address.com
  ```

Shell scripts

Motivation for using shell scripts:

- Automating simple tasks
- Scalable
- Easy to pass arguments

Syntax, via an example:

```
FOLDER=~/tmp/test_downloads/request_for_comments/
N=1149 # RFC number
filename=rfc$N.txt
mkdir -p $FOLDER
wget http://tools.ietf.org/rfc/$filename
mv $filename $FOLDER
```

Incorporating command line arguments can help to make a script far more flexible.

```
bash myscript.sh a ab abc
```

Makefiles

Makefiles allow a user to easily build/rebuild a set of files. This makes it particularly useful for compiled languages.

Makefiles are also able to capture dependencies between files so only the necessary files are re-built when some of the source files are changed. Also, independent commands can be run in parallel.

A makefile takes the following format:

```
target: source (what target depends on)
[tab]        commands
```

Given that *tabulatesin.c* calculates the sine of certain values, *plotfile.py* reads data from a file and plots a figure when run from the command line, the following makefile will compile the c program, creating an executable *tabulatesin*, whose output is redirected to *data.txt* and the figure is plotted using the python script.

```
figure: plotfile.py data.txt
        python plotfile.py data.txt

data.txt: tabulatesin
        ./tabulatesin > data.txt

tabulatesin: tabulatesin.c
        gcc tabulatesin.c -o tabulatesin -Wall -ansi
         ↪  -pedantic -lm

clean:
        rm data.txt dataplot.pdf tabulatesin
```

Note: These commands are for Linux; more complex makefiles may be able to handle different environments. The main changes required would be `tabulatesin.exe > da` and `del...` not `rm`.

git

Motivation for version control systems (VCS), such as git:

1. A clean way to back up programs: committing changes regularly allows one to roll back a program to a previous state with ease
2. Allows users to work from multiple machines
3. Supports collaboration

git is not natively installed in Windows. Installing git will usually install git from cmd and git bash, a Linux terminal for Windows, equipped with git bash.

git workflow

- Either `git init` This creates a repository in the current work directory.

  Or `git clone /path/to/repository`.

  This clones an online repository.

- `git add <file>`.

  Stages a file

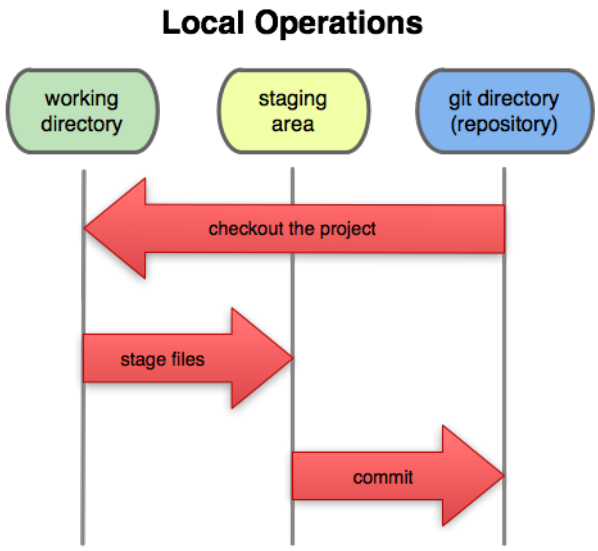

Figure 1: This figure illustrates the difference between the working directory, staging area and repository directory. One step further would be the remote repository which is achieved by pushing the git directory.

- `git commit -am "Initial commit"` Commits all files with a commit message "Initial commit".
- `git push origin master` With changes in the HEAD of your local working copy, this pushes the changes to your remote repository.
- `git remote add origin <server>` This connects your local repository to an a remote repository, allowing you to push your changes.

Symbolic Python

We can use symbolic algebra to go directly from high-level, compact form to explicit code, saving time, reducing errors and allowing high-level optimisations.

The snippet below is an example of a number ways SymPy, a Python implementation of symbolic methods, can be utilised:

```
import sympy as sy
from sympy.utilities.lambdify import lambdify
from sympy.utilities.codegen import codegen


def differentiate(syEquation, var):
    return sy.diff(syEquation, var)


def integrate(syEquation, var):
    return syEquation.integrate(var)


def solve(syEquation, var):
    return sy.solve(y, x)


x, y = sy.symbols("x y")
y = x**2 + 2 * x + 1

print(differentiate(y, x))
print(integrate(y, x))
print(solve(y, x))

yprime = differentiate(y, x)
ydiff = lambdify((x), yprime)
print(ydiff(-1))

code = codegen(('compute_ydiff', yprime), 'C', 'my_proj')
print(code)

for file in range(len(code)):
    with open(code[file][0], 'w') as fw:
        fw.write(code[file][1])
```

Cython

Cython is a programming language that acts a superset of Python, though demonstrating C-like performance. One can install the Cython package, and use a set a specific commands to automatically generate Cython code.

To do this, one requires a Python script and a setup script (also written in Python):

```
def primes_py(nb_primes):
    p = []
    n = 2
    while len(p) < nb_primes:
        # Is n prime?
        for i in p:
            if n % i == 0:
                break
            # If no break occurred in the loop
            else:
                p.append(n)
            n += 1
            return p
```

```
from distutils.core import setup
from Cython.Build import cythonize

setup(ext_modules=cythonize("example_py_cy.py",
                            annotate=True))
```

Upon running

```
python setup.py build_ext --inplace
```

from the command line, a number of files will be generated, including the .c file and an html for viewing the output. It hightlights code in different shades of yellow, based upon its dependence on python.

Part II

The C-programming language

File I/O

Function pointers

Function pointers point to a piece of code, not an address like a normal pointer.

Below is an example of a number of possible uses of function pointers:

```c
#include <stdio.h>
/* A normal function with an int parameter
and void return type*/
void fun(int a)
{
    printf("Value of a is %d\n", a);
}

void add(int a, int b)
{
    printf("Addition is %d\n", a + b);
}
void subtract(int a, int b)
{
    printf("Subtraction is %d\n", a - b);
}
void multiply(int a, int b)
{
    printf("Multiplication is %d\n", a * b);
}

int main()
{
    /* fun_ptr is a pointer to function fun() */
    void (*fun_ptr)(int) = &fun;
    void (*fun_ptr2)(int) = fun;
    void (*fun_ptrarr[])() = {add, subtract, multiply};

    /* The above line is equivalent of following two
       void (*fun_ptr)(int);
       fun_ptr = &fun;
    */
```

```
    /*Invoking fun() using fun_ptr*/
    (*fun_ptr)(10);

    /*Can also be invoked in this way*/
    fun_ptr2(10);

    (*fun_ptrarr[2])(1, 2);

    return 0;
}

/*
Output:
Value of a is 10
Value of a is 10
Multiplication is 2
*/
```

Useful algorithms

Fisher-Yates shuffle

This is a classic shuffling algorithm, that is more efficient than simply reproducing random number until an unseen one appears.

```
#include<time.h>

void randomize ( int arr[], int n )
{
    /*Use a different seed value so that we don't get same
     ↪
     result each time we run this program*/
    srand ( time(NULL) );

    /*Start from the last element and swap one by one. We
     ↪  don't
     need to run for the first element that's why i > 0*/
    for (int i = n-1; i > 0; i--)
    {
        /*Pick a random index from 0 to i */
        int j = rand() % (i+1);

        /*Swap arr[i] with the element at random index*/
        swap(&arr[i], &arr[j]);
    }
```

```
}
```