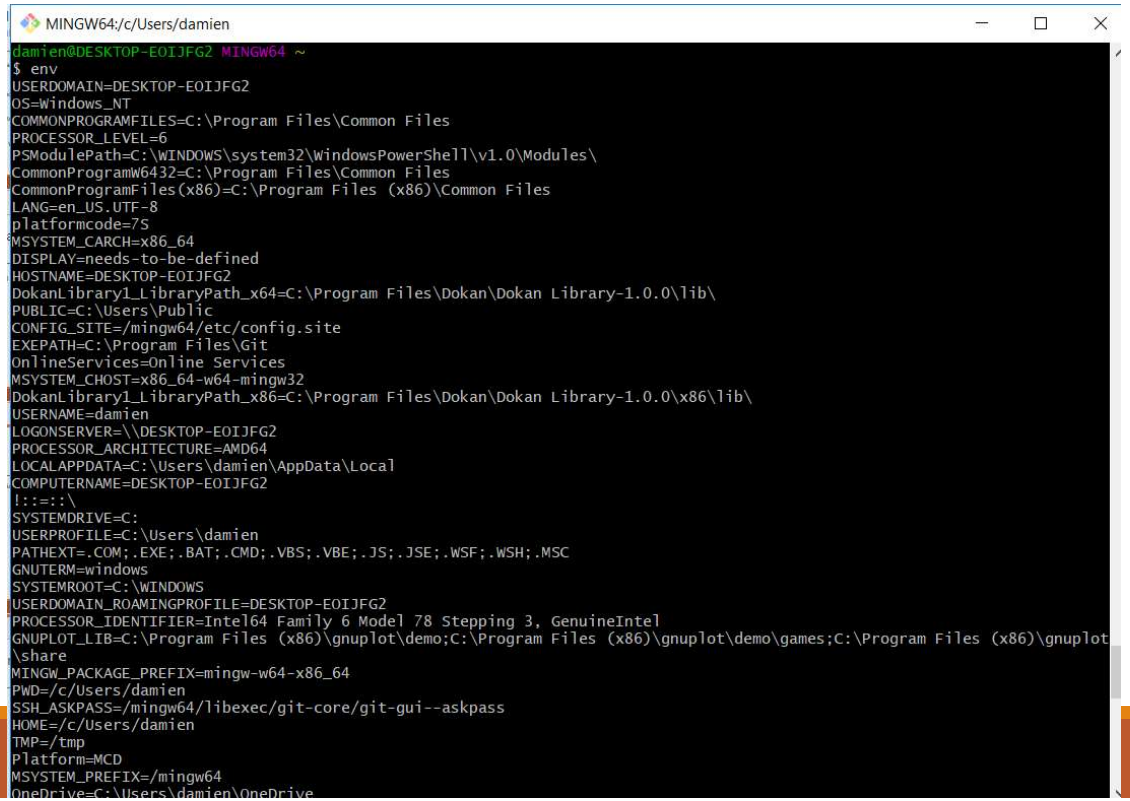
Lecture 2: Python I/O at the command line

- Continue working at the command line (environment)
- Scripts and Command line arguments
- Basic I/O in python scripts

Environmental variables

env

env shows a list of all currently defined environmental variables and what they are set to.



Changing them (with **export** or **setenv**) willy-nilly will most certainly break things. Many processes look at env to determine behavior.

To see the value of a particular variable you can just echo it, e.g:

echo $PATH

($PATH) What happens when you type ls?

```
$ echo $PATH
/c/Users/damien/bin:/mingw64/bin:/usr/local/bin:/usr/bin:/bin:/mingw64/bin:/usr/bin:/c/Users/damien
/bin:/c/ProgramData/Oracle/Java/javapath:/c/Program Files (x86)/Intel/iCLS Client:/c/Program Files/
Intel/iCLS Client:/c/WINDOWS/system32:/c/WINDOWS:/c/WINDOWS/System32/Wbem:/c/WINDOWS/System32/Windo
wsPowerShell/v1.0:/c/Program Files (x86)/Intel/Intel(R) Management Engine Components/DAL:/c/Program
 Files/Intel/Intel(R) Management Engine Components/DAL:/c/Program Files (x86)/Intel/Intel(R) Manage
ment Engine Components/IPT:/c/Program Files/Intel/Intel(R) Management Engine Components/IPT:/c/Prog
ram Files/Intel/IntelSGXPSW/bin/x64/Release:/c/Program Files/Intel/IntelSGXPSW/bin/win32/Release:/c
/Program Files/MiKTeX 2.9/miktex/bin/x64:/c/Users/damien/.dnx/bin:/c/Program Files/Microsoft DNX/Dn
vm:/c/Program Files (x86)/gnuplot/bin:/c/TDM-GCC-64/bin:/c:/c/WINDOWS/system32:/c/WINDOWS:/c/WINDOW
S/System32/Wbem:/c/WINDOWS/System32/WindowsPowerShell/v1.0:/c/Program Files/MATLAB/R2018a/bin:/cmd:
/c/Program Files/Intel/WiFi/bin:/c/Program Files/Common Files/Intel/WirelessCommon:/c/WINDOWS/Syste
m32/OpenSSH:/c/Users/damien/Anaconda3:/c/Users/damien/Anaconda3/Library/mingw-w64/bin:/c/Users/dami
en/Anaconda3/Library/usr/bin:/c/Users/damien/Anaconda3/Library/bin:/c/Users/damien/Anaconda3/Script
s:/c/Users/damien/AppData/Local/Programs/Python/Python35-32/Scripts:/c/Users/damien/AppData/Local/P
rograms/Python/Python35-32:/c/Users/damien/AppData/Local/Programs/MiKTeX 2.9/miktex/bin/x64:/c/User
s/damien/AppData/Local/Microsoft/WindowsApps:/usr/bin/vendor_perl:/usr/bin/core_perl
```

Almost all shell commands are just individual programs. When you type “ls” it looks through each dir in the $PATH variable until it finds an executable file named ls

/c/Users/damien/bin ✗
/mingw64/bin ✗
/usr/local/bin ✗
/usr/bin ✓
/bin …

```
damien@DESKTOP-EOIJFG2 MINGW64 ~
$ which ls
/usr/bin/ls
```

.bashrc file

Whenever a bash shell is initiated it executes the .bashrc script (if it exists in your home directory). This script is commonly used to change environmental variables or define aliases.

aliases are defined keyboard shortcuts for commands

```
damien@DESKTOP-EOIJFG2 MINGW64 ~
$ alias
alias ipython='winpty ipython.exe'
alias ll='ls -l'
alias ls='ls -F --color=auto --show-control-chars'
```

winpty is used to fix some compatibility issues between bash and windows.

This is commonly used as a way to alter your environmental variables or issue commands associated with your environment.

use nano to create a .bashrc file in your home directory

nano .bashrc

```
  GNU nano 2.9.3            .bashrc                 Modified

alias python='winpty python'
alias gnuplot='winpty gnuplot'
alias cp='cp -i '
```

starting a new bash shell should automatically define these new aliases.

```
damien@DESKTOP-EOIJFG2 MINGW64 ~
$ alias
alias cp='cp -i '
alias gnuplot='winpty gnuplot'
alias ipython='winpty ipython.exe'
alias ll='ls -l'
alias ls='ls -F --color=auto --show-control-chars'
alias python='winpty python'
```

Adding to your path

If you cannot execute a command at the command line, it is probably not in your path.

export PATH=$PATH:newdir (to add newdir to your PATH)

```
damien@DESKTOP-EOIJFG2 MINGW64 ~
$ chrome
bash: chrome: command not found
```

to get chrome working at the command line you could add the following line to your .bashrc script

```
damien@DESKTOP-EOIJFG2 MINGW64 ~
$ export PATH=$PATH:"/c/Program Files (x86)/Google/Chrome/Application"
```

new PATH variable

Important! This is your current PATH variable. Without this none of your other commands will work.

directory to add to your path.

Writing a script

Writing a script

Create a Hello World! Bash script using nano

Tells the shell which interpreter to use when running the script

```
  GNU nano 2.9.3                    hello.sh

#!/bin/bash
echo "Hello world! from $0"
```

In linux, the file must be “executable” for you to run it directly (./hello.sh)

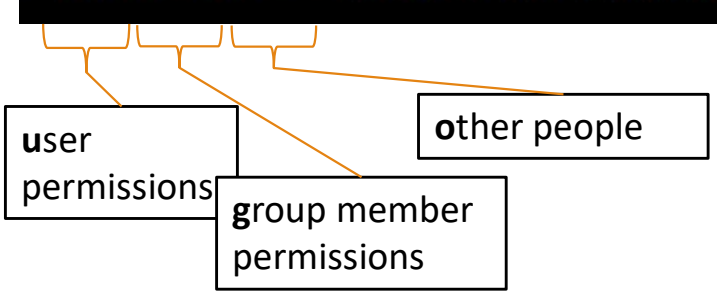
can be run with
./hello.sh
sh hello.sh
or bash hello.sh

running a script with **bash –x** script.sh is useful for debugging.

permissions

ls –l (-l tells ls to give more details, including showing the permissions)

```
$ ls -l
total 2
-rwxr-xr-x 1 damien 197121 104 May 23 16:22 hello.sh*
-rw-r--r-- 1 damien 197121  20 May 23 16:31 newfile.sh
```



permissions: r ~ read permission
w ~ write permission
x ~ execute permission

to add execute permissions to newfile:
chmod *a+x* newfile.sh (for all)
chmod *u+x* newfile.sh (for user)
chmod g+x newfile.sh (for group members)
chmod o+x newfile.sh (for others)

can remove permissions with
chmod a-x newfile.sh

Command line arguments

command line args can be used to specify behavior/input files/output files. (e.g., -l is a command line arg for ls)

```
  GNU nano 2.9.3                                hello.sh

#!/bin/bash
echo "Hello World! from $0"

if [ $1 ]
        then
        echo "Your first command line argument was $1!"
fi
```

Command line arguments can be referred to in your script with $0, $1, $2, etc.

$0 = name of script

$1 = 1st command line argument

$2 = 2nd command line argument, etc.

can loop over command line args with

```
for i in $@;do something;done
```

A little more useful

```
  GNU nano 2.9.3                          comm

#!/bin/bash
i=0
for var in `cat $1`
        do mkdir a$i
        sed "s/alph.*/alpha = $var/g" INPUT>a$i/INPUT
        ((i++))
done
```

What does running this script do??

./comm nums

For context, here are the other files in directory

```
$ cat nums
2.3
4.1
6.7
2.2
1.6
```

```
$ cat INPUT
This is my example input file
zeta = True
beta = 22.1
ALGO = 45
alpha = 1.6
POS={12.3,2.1,3.2}
```

Writing a python script

```
  GNU nano 2.9.3                                        hello.py

print("Hello World!")
```

can be executed with python hello.py

```
$ python hello.py
Hello World!
```

To make it act as standalone executable, we can put in the interpreter line.

```
  GNU nano 2.9.3                 /c/Users/damien/bin/hello

#!python
print("Hello World!")
```

Since it is in my path, I can execute it just like a command:

```
$ hello
Hello World!
```

Command Line arguments

```
  GNU nano 2.9.3                          example1

#!python
import sys

print ("This is the name of the script: ", sys.argv[0])
print ("Number of arguments: ", len(sys.argv))
print ("The arguments are: " , str(sys.argv))
```

output

```
$ example1 arg1 arg2
This is the name of the script:  C:/Users/damien/bin/example1
Number of arguments:  3
The arguments are:  ['C:/Users/damien/bin/example1', 'arg1', 'arg2']
```

sys.argv[0] (name of file being executed)
sys.argv[1] (1st command line argument)
sys.argv[2] (2nd command line argument), etc.

Making a simple python plotting program

```
  GNU nano 2.9.3                                  pplot.py

#!python
import numpy as np
import matplotlib.pyplot as plt
import sys

with open(sys.argv[1],"r") as inputfile:
        vals=np.loadtxt(inputfile)  #returns numPy array

plt.plot(vals[:,0],vals[:,1])
plt.show()
```

#vals[:,0] is 1st column of data
#vals[:,1] is 2nd column of data

numpy.**loadtxt**(*fname, dtype=<class 'float'>, comments='#', delimiter=None, converters=None, skiprows=0, usecols=None, unpack=False, ndmin=0, encoding='bytes', max_rows=None*)¶ [source]

numpy.**savetxt**(*fname, X, fmt='%.18e', delimiter=' ', newline='\n', header='', footer='', comments='# ', encoding=None*)

numPy loadtxt and savetxt:

for fast reading/writing of simply formatted files (each line has same # of values)

Activity 2: make standalone python script with command line arguments

1) get the data file named "data.csv" located at 74.69.18.77
   username: class password: phys2962
   (hint: the ***find*** command is very useful,
   https://www.linode.com/docs/tools-reference/tools/find-files-in-linux-using-the-command-line/)

2) Write a standalone python script which can read-in a data file (*inputfname*) containing columns of comma separated values. It should add two selected columns (*num1* and *num2*) together and write the sum as a single column to a selected output file (*outputfname*).

The script should take 4 command line arguments:

Usage: ./script.py *inputfname outputfname num1 num2*

(note: elements of sys.argv are strings, num1 and num2 will need to be cast as integers).

3) Run the script on columns 1 and 3 of data.csv (./script.py data.csv outfile 1 3).
   ** note vals[:,0] is 1st column, vals[:,1] is 2nd column, etc.
4) Upload **your script** and the resulting **outfile** to the LMS.

Note: Probably the fastest way to do this in real life would be at the command line with sed 's/,/ /g' data.cvs|gawk '{print $1+$3}'>outfile