EEB 177 Lecture 6

Topics

- Permissions
- Intro to Python

Hacking Sessions

Tonight from 6-8 in LS 3209

Homework #2

Due this Friday (tomorrow) by 5:00 PM

Projects

Project ideas are due Friday, Feb 3rd.

Preliminaries

- Start **gedit**: `$ gedit` and save the file "classwork-Thursday-1-26.txt" to your class-assignments directory
- push this to your remote repository
- you can write answers to today's exercises in this file.

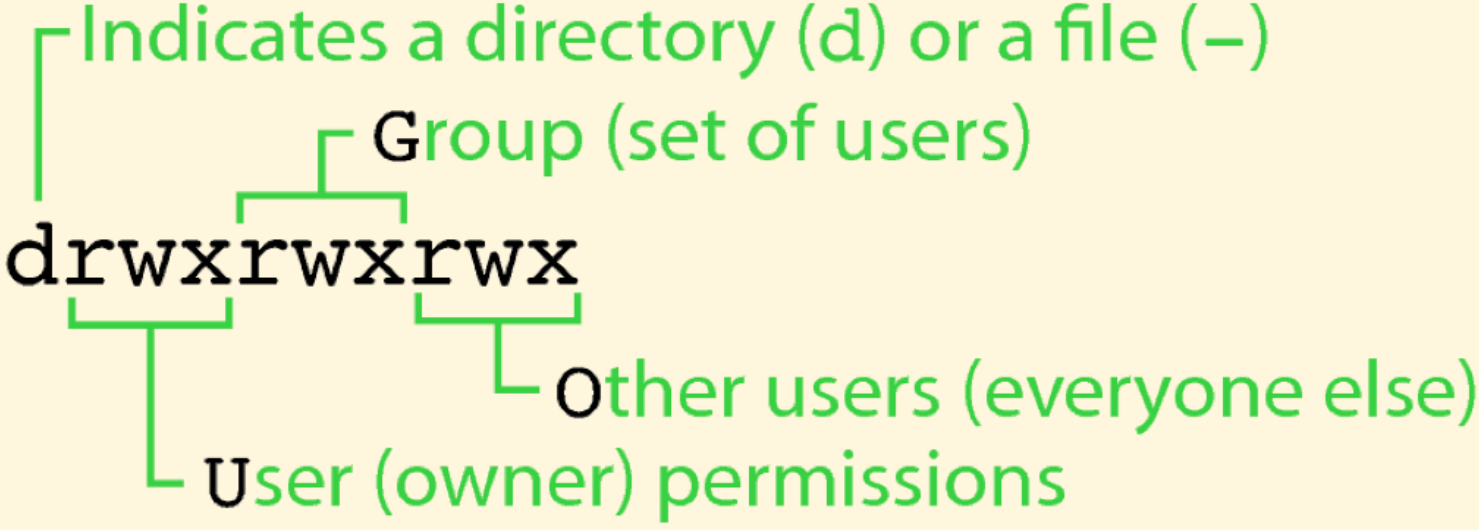
Permissions

In Unix, each file and directory has an attribute that determines who can read (r), write (w), execute (x), or do nothing (-) to a file. There are three categories of file users

- owner
- group (set of users)
- everyone else

you can see permissions with `ls -l`

permissions structure



`chmod` and `chown`

These commands change permissions and ownwership (`u` , `g` , or `o` for user, group or other).

```
touch permissions.txt
ls -l
```

```
-rw-rw-r-- 1 eeb-177-student eeb-177-student 0 Jan 24 07:5
```

```
chmod u=rwx permissions.txt
ls -l
-rwxrw-r-- 1 eeb-177-student eeb-177-student     0 Jan 24
```

notice that the user may now execute this file.

you can also add and remove permissions for a user with `+` and `-`.

```
chmod g+w,u+x permissions.txt
ls -l permissions.txt
-rwxrw-rw-
```

Add write permissions for all users.

Remove read, write, and execute permission from others

your text editor that will execute a series of shell commands that you have already learned.

open up gedit and type the following lines:

```
#! /bin/bash
ls -la
echo "Above are the directory listings for this folder"
pwd
echo "right now it is :
date
```

save this file as dir.sh

Paths

there are two standard locations for programs-- `/usr/bin` and `/bin`

use `ls` to see what is in them

Creating a scripts directory and adding it to the path

we want a single working copy of each program on our machines so we need to make sure the shell searches for our programs….

- go to your home directory
- create a directory called `scripts`
- to add the scripts directory to the path, open the `.bash_profile` file in gedit
- add this line (exactly) `export PATH="$PATH:$HOME/scripts"`
- exit and save

Now we have created a program we would like to run and created a path to the scripts directory. What else do we need to do?

hint: where is `dir.sh` right now?

hint: what permissions do we need to execute a script?

the shebang (#!)

`#!` is called the shebang--it means that all following contents of script will be sent to the program following the shebang

`#! /bin/bash` sends it all to bash

remember,new scripts are not executable w/o changing permissions

checking permissions

- cd ~/scripts
- check permission with ls -l
- add permission to execute with chmod u+x

try running your program from different directories. Does it work? Why?

exercise

Add your scripts directory to your remote repository. You will need to

- `git init` in your scripts directory
- add your script
- commit your script
- create a remote repo on github
- copy and paste the command lines from the remote repo after you create it.

```
git remote add origin https://github.com/michaelalfaro/eeb
git push -u origin master
```

you will be using your scripts directory throughout the quarter, so make sure this repo is working

Programming languages

- There are over 2000
- There is no perfect language for all tasks
- You are already learning several: regular expressions, python, R
- This class does not cover fast, compiled languages like C. Useful for heavy computational tasks

Why `Python`?

- easy to teach
- readable
- powerful string manipulation, web scraping, and other capabilities
- helps enforce good programming practices

Getting started with python

- Python should already be installed on your VM

```
$ ipython
IPython 2.2.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's featur
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for ex
In [1]: import scipy
```

to exit ipython: CTRL-D CTRL-D to exit

IPython interactive mode

You can use python and IPython from the command prompt.

```
In [1]: 2 + 2
Out[1]: 4

In [6]: 2 > 3
Out[6]: False

In [11]: "I'm fine, " + "thank you"
Out[11]: "I'm fine, thank you"
```

try it out....

Python operators

```
+ Addition
- Subtraction
* Multiplication
/ Division
** Power
% Modulo
// Integer division == Equals
!= Differs
> Greater
>= Greater or equal &, and Logical and |, or Logical or
!, not Logical not
```

Variables

You typically manipulate variables in programming languages.

```
In [12]: x = 5
In [13]: x
Out[13]: 5
In [14]: x + 3
Out[14]: 8
In [15]: y = 8
In [16]: x + y
Out[16]: 13
In [17]: x = "My string"
In [18]: x + " is now longer"
Out[18]: "My string is now longer"
In [19]: x + y
TypeError: cannot concatenate "str" and "int" objects
```

Python recognizes variable types and methods for converting among them

```
In [20]: x
Out[20]: "My string"
In [21]: y
Out[21]: 8
In [22]: x + str(y)
Out[22]: "My string8"
In [23]: z = "88"
In [24]: x + z
Out[24]: "My string88"
In [25]: y + int(z)
Out[25]: 96
```

Collections

Python has variables that are **collections** of other objects. **lists** are collections of ordered values and are defined by `[]`.

```
# Anything starting with # is a comment
In [26]: MyList = [3,2.44,"green",True]
In [27]: MyList[1]
Out[27]: 2.44
In [28]: MyList[0] # NOTE: FIRST ELEMENT -> 0
Out[28]: 3
In [29]: MyList[3]
Out[29]: True
In [30]: MyList[4]
IndexError: list index out of range
In [31]: MyList[2] = "blue"
In [32]: MyList
Out[32]: [3, 2.44, "blue", True]
```

note that indexing in python starts at 0.

{.smaller}

append, **sort**, and **count** are methods that work on lists

```
In [33]: MyList[0] = "blue"
In [34]: MyList
Out[34]: ["blue", 2.44, "blue", True]
In [35]: MyList.append("a new item")
In [36]: MyList
Out[36]: ["blue", 2.44, "blue", True, "a new item"]
In [37]: MyList.sort()
In [38]: MyList
Out[38]: [True, 2.44, "a new item", "blue", "blue"]
In [39]: MyList.count("blue")
Out[39]: 2
```

More list operations

Try these

```
In [71]: apes = ["Homo sapiens", "Pan troglodytes", "Gori
In [72]: #how long is list?
In [73]: len(apes)
Out[73]: 3
In [74]: #index elements with brackets
In [75]: apes[1] # what element will this be
Out[75]: 'Pan troglodytes'
In [76]: apes.reverse() #lists are ordered
In [77]: apes[0]
Out[77]: 'Gorilla gorilla'
#if you know the element but do not know position, use ind
In [78]: apes.index("Homo sapiens")
Out[78]: 2
#get the last element of any list with [-1]
In [79]: apes[-1]
Out[79]: 'Homo sapiens'
```

Subsetting lists

You can make a new list by telling python the elements of the original list to include

```
ranks = ["kingdom","phylum", "class", "order", "family"]
In [81]: ranks
Out[81]: ['kingdom', 'phylum', 'class', 'order', 'family']
lower_ranks = ranks[2:5]
In [83]: lower_ranks
Out[83]: ['class', 'order', 'family']
```

Note how python indexing works:numbers are **inclusive** at the start and **exclusive** at the end.

Appending to lists

Try this

```
apes = ["Homo sapiens", "Pan troglodytes", "Gorilla gorill
print("There are " + str(len(apes)) + " apes")
apes.append("Pan paniscus")
print("Now there are " + str(len(apes)) + " apes")
```

append() changes the variable in place!

+ and extend

Try this

```
apes = ["Homo sapiens", "Pan troglodytes", "Gorilla gorill
monkeys = ["Papio ursinus", "Macaca mulatta"]
primates = apes + monkeys
print(str(len(apes)) + " apes")
print(str(len(monkeys)) + " monkeys")
print(str(len(primates)) + " primates")

platyrrhines = ["Cebus", "Sapajus", "Aotus"]
monkeys.extend(platyrrhines)
print monkeys
```

{.smaller}

Dictionaries are collections of key-value pairs. Very useful for data without a natural order. Defined by `{}`.

```
In [1]: GenomeSize={"Homo sapiens": 3200.0,
  "Escherichia coli": 4.6, "Arabidopsis thaliana": 157.0}
In [2]: GenomeSize
Out[2]:
{"Arabidopsis thaliana": 157.0,
 "Escherichia coli": 4.6,
 "Homo sapiens": 3200.0}
```

You can access the value of a dictionary by suppling the key for that pair.

```
In [3]: GenomeSize["Arabidopsis thaliana"]
Out[3]: 157.0
In [4]: GenomeSize["Saccharomyces cerevisiae"] = 12.1
In [5]: GenomeSize
Out[5]:
{"Arabidopsis thaliana": 157.0,
 "Escherichia coli": 4.6,
 "Homo sapiens": 3200.0,  "Saccharomyces cerevisiae": 12.1
```

Adding keys and changing values in dictionaries

```
# ALREADY IN DICTIONARY!
In [6]: GenomeSize["Escherichia coli"] = 4.6
In [7]: GenomeSize
Out[7]:
{"Arabidopsis thaliana": 157.0,
 "Escherichia coli": 4.6,
 "Homo sapiens": 3200.0,
 "Saccharomyces cerevisiae": 12.1}
In [8]: GenomeSize["Homo sapiens"] = 3201.1
In [9]: GenomeSize
Out[9]:
{"Arabidopsis thaliana": 157.0,
 "Escherichia coli": 4.6,
 "Homo sapiens": 3201.1,
 "Saccharomyces cerevisiae": 12.1}
```

Creating dictionaries

You will often create a dictionary and then populate it. Try this!

```
enzymes = {}
enzymes['EcoRI'] = r'GAATTC' # r before the string tells p
enzymes['AvaII'] =  r'GG(A|T)CC'
enzymes['BisI'] =  r'GC[ATGC]GC'
enzymes.keys()
enzymes.values()
```

You can use **zip()** to turn two lists into a dictionary

```
keys = ('name', 'age', 'food')
values = ('Monty', 42, 'spam')
zip(keys, values) #makes a list of tuples
my_new_dict = dict(zip(keys, values))
my_new_dict
```

tuples contain sequences that are immutable and are defined by **()**.

```
In [12]: FoodWeb=[("a","b"),("a","c"),("b","c"),("c","c")]
In [13]: FoodWeb[0]
Out[13]: ("a", "b")
In [14]: FoodWeb[0][0]
Out[14]: "a"
# Note that tuples are "immutable"
In [15]: FoodWeb[0][0] = "bbb"
TypeError: "tuple" object does not support item assignment
In [16]: FoodWeb[0] = ("bbb","ccc")
In [17]: FoodWeb[0]
Out[17]: ("bbb", "ccc")
```

Use tuples when order matters.

sets are lists without duplicate elements

```
In [1]: a = [5,6,7,7,7,8,9,9]
In [2]: b = set(a)
In [3]: b
Out[3]: set([8, 9, 5, 6, 7])
In [4]: c=set([3,4,5,6])
In [5]: b & c
Out[5]: set([5, 6])
In [6]: b | c
Out[6]: set([3, 4, 5, 6, 7, 8, 9])
In [7]: b ^ c
Out[7]: set([3, 4, 7, 8, 9])
```

The operations are: Union | (or); Intersection & (and); symmetric difference (elements in set b but not in c and in c but not in b), ^; and so forth.

You can concatenate similar collection elements with +

```
In [1]: a = [1, 2, 3]
In [2]: b = [4, 5]
In [3]: a + b
Out[3]: [1, 2, 3, 4, 5]
In [4]: a = (1, 2)
In [5]: b = (4, 6)
In [6]: a + b
Out[6]: (1, 2, 4, 6)
In [7]: z1 = {1: "AAA", 2: "BBB"}
In [8]: z2 = {3: "CCC", 4: "DDD"}
In [9]: z1 + z2
     ---------------------------------------------
     TypeError  Traceback (most recent call last)
     ----> 1 z1 + z2
TypeError: unsupported operand type(s) for +: "dict" and '
```

Python programming best practices

We will try to instill as many standard practices as we can at the beginning.

- Wrap lines so that they are less than 80 characters long. You can use parentheses () or signal that the line continues using a "backslash" .
- Use 4 spaces for indentation, no tabs.
- Separate functions using a blank line.
- When possible, write comments on separate lines.
- Use docstrings to document how to use the code, and comments to explain why and how the code works.

best practices continued

- Naming conventions:
- _internal_global_variable
- a_variable
- SOME_CONSTANT
- a_function
- Nevercallavariablel 1 or O or o
- use two or more letters for variables: xx better than x
- Use spaces around operators and after commas:
  a = func(x, y) + other(3, 4).