UNIVERSITY OF CALGARY
FACULTY OF SCIENCE
Department of Physics and Astronomy

Physics Teaching Laboratory Website Manual

Version: 0.5

Contents

1.1 Lab Information Hub Website

1.1.1 Introduction

The purposes of the pjl website is to be the central information hub for the educational physics labs. It is a base of knowledge from which the department can work collaboratively on building the future of education physics labs.

Guiding Principles

- Documents must be accompanied by code required to generate the document.

- Documents posted in PDF format.

- Only documents that have been deployed are to be posted.

- **NOT FINISHED**

NEED CONTENT

1.1.2 Goals

- Central

- Accessible

- Secure

- Transferable

NEED MORE CONTENT

1.1.3 How Instructors Can Use the Hub

NEED MORE CONTENT

1.1.4 How Technicians Can Use the Hub

NEED MORE CONTENT

1.2 Definitions

Development Space: A folder (/dev) inside the the Testing Version root directory that contains all files that are edited when adding content to the repository or the inventory. It also contains tools used for

adding content. **LOOK INTO IF I CAN MOVE ALL OTHER FILES THAT CAN BE EDITED WHEN ADDING CONTENT HERE**.

Inventory: Complete list of all equipment in the educational labs. This information is organized in a xml file (/data/equipmentDB.xml), and references photos and manuals stored in an equipment folder (/staffresources/equipment).

Live Version: The most current version of the Physics Educational Laboratory Information Hub available to the public. This version lives on the apache web-server.

Repository: Complete collection of all lab document files (/data/repository), and a xml file (/data/labDB.xml) that contains organizational information about the files in the repository.

Root Folder: A folder that contains all file related to the Physics Educational Laboratory Hub (pjl-web). All references to a folder or file are made relative to the root folder.

Testing Version: A version of the Physics Educational Laboratory Information Hub that needs to have recent edits tested. This version lives on the development server.

Web-Server: A linux (watt.pjl.ucalgary.ca) that is running apache2.

2 Process for Editing the Website

2.1 Introduction

- Live Version - Lives on Watt - What the public sees.

- Testing Version - Lives in pjl-web on Slug - Used to review changes before the are committed.

- Development Space - live in pjl-web/dev on Slug - Where the changes are made.

2.2 Overview of Editing Process

Follow these steps for making all updates to website. The purpose of each step has been explained in Sections 2.3 - 2.6.

1. Sync Testing Version with Live Version. See Section 2.3.

2. Make changes to the website. See Section 2.4.

3. Test the changes. See Section 2.5.

4. Sync Live Version with Testing Version. See Section 2.6.

2.3 Sync Testing Version with Live Version

The equipment data base can be modified using the live version of the website, therefor it is important to check to make sure that any changes made to the equipment database by use of the live website have been applied to the development space before more changes are made.

The script "liveUpdate.py" (Listing 6.1.1) will compare the files "data/equipmentDB.xml" on the live server with the one in the development space. If the live version in newer it will replace the version in the development space with the one on the live server.

To sync run...

sudo ./liveUpdate.py

For more information on how the script works run...

```
sudo ./liveUpdate.py −−help
```

NEED TO WRITE CODE INTO SCRIPT THAT WILL REPLACE /DEV/*.XML WITH /DATA/*.XML

2.4 Make Changes to the Website

All changes to the website should be made on the development space on slug (/usr/local/master/pjl-web). The only exception to this rule is that the equipment database equipmentDB.xml can be modified live by using the inventory website in edit mode, as mentioned in section 2.3. It is to only make changes in one place at a time. Do not make changes using the live website if changes are being made to the development space.

For specifics on how to make changes to the repository see Chapter 3.

For specifics on how to make changes to the equipment inventory see Chapter 4.

For specifics on how to make changes to the schedules see Section 5.1.

For specifics on how to make changes to the safety documents see **NEED THIS SECTION**

For specifics on how to make changes to the standard procedure documents see **NEED THIS SECTION**

2.5 Testing the Changes

Inside the folder pjl-web/date are the most current version of the repository xml file (labDB.xml) and the equipment xml file (equipmentDB.xml). Inside the same folder are the past nine version of each xml file. For example there exists files labDB-0.xml to labDB-8.xml where labDB-0.xml is the newest past version and labDB-8.xml is the oldest.

If changes have been made to the document repository run...

```
sudo ./repWheel.py
```

This script will remove the oldest version, labDB-8.xml, shuffle the rest of the version down one, and then replace the current version with the development version (pjl-web/dev/labDB.xml).

Similarly if changes have been made to the inventory run...

```
sudo ./eqWheel.py
```

Now start a local web-server to test out the Testing Version, buy running the following code while in the pjl-web folder

```
python −m SimpleHTTPServer 8000
```

The Testing Version can be view by opening a web browser and going to the URL "localhost:8000". Confirm that changes were made as expected.

The links for the website can be manually tested by running.

```
./linkCheck.py
```

Check into any result that does not return "STATUS: 200"

2.6   Sync Live Version with Testing Version

Once the changes to the development space have been made and tested the changes can be pushed to the web-server.

Update the live version by running...

To sync run...

```
sudo ./liveUpdate.py
```

For more information on how the script works run...

```
sudo ./liveUpdate.py −−help
```

3   Repository Content

3.1   Introduction

"Experiment Documents" are a collection of documents used by students in the educational labs, as well as all supporting documents. **ADD SOME EXAMPLES**

3.1.1   Criteria for adding document

Documents can only be added to the repository if they meet the following criteria.

1. The files include the pdf given to students to be used in their course work.

2. All files need to generate the pdf are included.

3.1.2   Directory structure

At top most level is a folder called "repository" that contains all experiment related documents.

At the second level all the files are organized by lab experiment. Each experiment has a folder that is labeled with a naming scheme where the first four characters are the unique identifier number, followed by the name of the lab. The lab name should be descriptive of the experiment itself. In this folder is also a folder called "Support_Docs" that contains any documents useful for the experiment, but not actually used to generate the student document.

At the third level files are organized into versions. Each folder follows a naming scheme where the the first four characters are the unique lab identifier number, followed by "PHYS" followed by the Course Number followed by a two character semester identifier, followed by the year. Each folder contains all the file used to generate the pdf given to students in the course, semester, and year as identified in the folder label.

Directory structure sample.
```
/repository
└── 0072-Nuclear-Decay
    └── 0001-PHYS123FA2017
```

```
        ├── lab.tex
      ├── photo.jpg
      ├── student.pdf
  └── Support_Docs
```
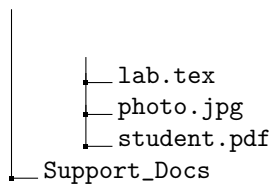
3.2   Editing Experiment Documents

3.2.1   Introduction

All changes should be made to the fill with the word "FULL" in the title. This is one document should contain everything needed to compile the student version and the TA version of an experiment. Different version of a experiment are compiled using the script pjldoc.py **REFERENCE TO HOW TO USE THIS SCRIPT**

3.2.2   Repository xml template

```
<Labs>
    <Lab labId="0001">
        <Name />
        <Disciplines>
            <Discipline />
            ...
        </Disciplines>
        <Topics>
            <Topic />
            ...
        </Topics>
        <Versions>
            <Version>
                <Path />
                <Semester />
                <Year />
                <Course />
                <Directory />
            </Version>
            ...
        </Versions>
        <Equipment>
            <Item id="0001">
                <Name />
                <Amount />
            </Item>
            ...
        <Equipment />
        <Type />
        <SupportDocs>
            <Doc>
                <Name />
                <Path />
            </Doc>
            ...
        </SupportDocs>
        <Software>
            <Name />
            ...
        </Software>
    </Lab>
    ...
</Labs>
```

3.3 Preparing Experiment Document Content

3.3.1 Introduction

- All tex in one file (lab and companion guide)

- standard preamble file

- pjldoc script for compiling documents

- documents prepared with a root directory of "under-construction"

- document editing timeline

3.3.2 Preparing the Documents

The following instructions where made specifically for physics 325 in winter 2019. Adjust the names for course and semester.

1. Create folder for course named in the form similar to **PHYS325WI2019**.

2. Inside course folder place a sub folder for each lab named in the form **0078-PHYS325WI2019**.

3. Inside each lab folder there should be...

   - tex file which include student version and companion guide. Name in the form **NAME-FULL-WI2019.tex All edits are made to this file**
   - Any documents referenced in the tex file which are need for compiling
   - Folder called **Support_Docs** that contain any important documents that are not needed to compile pdfs, such as sample data.
   - File called **standard-preamble.tex**

4. Inside main course folder make a text file called **physics325-lab-order**. Inside this folder list the id numbers of each experiment in the order it should appear in the manual. Be careful not to leave a black line at the end of the file.

3.3.3 Compiling Manuals and PDFs

Any set of documents that have been prepared as laid out in section 3.3.2 can be compiled in different configurations with the script **pjldoc.py**.

Help Section for pjldoc.py

A full list of flags and uses of the pjldocs.py script can be found by running

pjldoc.py −−help

Generating Student PDFs

To compile all of the student PDFs

pjldoc.py PHY325WI2019 −s −c −i 0

To compile individual PDF of the second lab listed in physics325-lab-order

pjldoc.py PHY325WI2019 −s −c −i 2

Generating Companion Guides for TA

To compile all of the Companion Guide PDFs

pjldoc.py PHY325WI2019 −t −c −i 0

3.4   Adding new version of an existing lab

Note that a version can only be added once, so make sure that everything outlined below is as desired before proceeding to step 4

1. Make a folder that will contain all file relating to the experiment to add. (ex "0078-PHYS325WI2019" is a suggested name for a folder that would be used to add lab 0078, used in physics 325, for the Winter 2019 semester.)

2. Inside the lab folder make a folder called "Support_Docs". This folder name is not optional because it will be reference in the scripts used for document and website maintenance.

3. In the main folder place. 3.1

   - Main tex file. (ex, Rutherford-Scattering-FULL-WI2019.tex)
   - Student tex file. (ex, Rutherford-Scattering-ST-WI2019.tex)
   - TA guide if it exists. (ex, Rutherford-Scattering-CG-WI2019.tex)
   - PDF of student version of lab. (ex. Rutherford-Scattering-ST-WI2019.pdf)
   - Any file needed to compile the student version of the pdf. (ex, setup-photo.jpg or standard-preamble.tex)
   - Any file that is particular this version of an experiment. (ex template-WI2019.xlsx)
   - Place any general documents into the folder called "Support_Docs". (ex, Interesting-Paper.pdf)

4. Run the command.

   sudo ./repositoryEdit −−add

   Example *I/**O*** from script when adding experiment version.

   > *Enter lab ID number:*
   > **0087**
   > *Adding version to "Rutherford Scattering"*

```
/0078-PHYS325WI2019
├── Rutherford-Scattering-FULL-WI2019.tex
├── Rutherford-Scattering-ST-WI2019.tex
├── Rutherford-Scattering-CG-WI2019.tex
├── Rutherford-Scattering-ST-WI2019.pdf
├── setup-photo.jpg
├── standard-preamble.tex
├── template-WI2019.xlsx
├── Support_Docs
    └── Interesting-Paper.pdf
```

Figure 3.1. Directory structure and sample contents.

Enter absolute path for directory containing lab:

/home/pgimby/labs/under-construction/PHYS325WI2019/0078-PHYS325WI2019

Enter the name of the student version pdf:

Rutherford-Scattering-ST-WI2019.pdf

Enter course number:

325

Enter semester:

Winter

Year

2019

Would you like to edit the equipment list for this lab? y/N

n

Would you like to edit the software list for this lab? y/N

n

Would you like to edit the disciplines list for this lab? y/N

n

Would you like to edit the topics list for this lab? y/N

n

Is this information correct? N/y:

y

Note that the lists of equipment, software, disciplines, and topics can be edited here if desired. For more information on the see **NEED REFERENCE**

5. Sync live version. See section **??**

3.5   Adding a new lab to the repository

Before beginning ensure that all equipment used in the new experiment are in the lab inventory, and have equipment ID number.

1. Create a folder for the new lab (example "new-lab-folder"), and place all files for generating student pdf, and the student pdf in new-lab-folder

2. Inside new-lab-folder make a directory called "Support_Docs", and put all documents relevant to lab, but not needed for generation of pdf into it. This might include research papers, sample data, Excel spreadsheets, etc.

```
sudo ./repositoryEdit −n
```

The command can also be run in test mode by executing...

```
sudo ./repositoryEdit −n −t
```

The script will now take you through several steps to gather the information needed to properly add this new lab to the repository. There are several safeties built into the code, but there will be a request to review the input information and confirm that it is correct. Please take time at this point to carefully review metadata entered.

The disciplines, topics, and software entries must align with the master list contained in /data/valid-Diciplines.txt, /data/validTopics.txt, and /data/validSoftware.txt. New items must be added to these master lists before they can be added to a lab.

- Name, **Name as to be Seen on Website** *Use Standard Title Capitalize Convention.*
- Type, **Type** *Must be either Lab or Labatorial.*
- Disciplines, **Discipline1, Discipline2** *Disciplines must comma separated be taken from the approved list* **Need location of this list.**
- Topics, **Topic1, Topic2** *Topics must comma separated be taken from the approved list* **Need location of this list.**
- Semester, **Semester** *Winter, Spring, Summer, or Fall*
- Year, **Year** *: Four digits.*
- Course, **Course Number** *Three digit number corresponding to the course the experiment was used in.*
- Equipment, **equipID-(Amount)-[alternate equipID], equipID-(Amount)** *equipID is four digit code of equipment in inventory, Amount is how many are needed, alternate ID is the four digit code of equipment in inventory that can be used if the primary unit is not available. IDs amounts and alternate IDs separated by "-", and items in equipment list separated by ","*
- Software, **Software1, Software2** *Name of all software needed. Must be software from the list of supported software* **Need location of this list**
- PDF, **PDF exact Name** *This needs to be the exact name of the student pdf*

4 Inventory Content

4.1 Inventory structure

Each item in the inventory should have a unique identifier number, and a unique name. An item can be either a stand alone item, or a kit. If the item is a kit it will need to include a list of the items the kit. If only part of the kit are needed for an experiment the repository xml can reference those items by creating a equipment tag in the labDB.xml that has the id number for the kit, but the name for the individual item(s). Each item has a place for any number of manuals, and one picture.

All changes outlined in this section are made to the development side of the website. Once all changes have been made, and are satisfactory, the live version of documents must be update and the web server must be updated.

4.2 Adding New Equipment

From the /dev/python-tools folder, the command,

```
./equipmentEdit.py −n
```

A prompt will appear that will ask for information regarding the new item. Enter all the information available. It is ok leave some fields blank just as long as there is a name. Once all the available information is added, the user will be asked to confirm the information entered. The new item entry will then go through a validation process to ensure that the name is unique. If everything checks out the new item will be added to the /dev version of the equipmentDB.xml file.

Note: To add Greek letters enter them as (ex. {Omega} or {mu}).

4.3 Adding Photos of Equipment

When taking photos note that they will require editing before they are added to the database. The final version of photo will be square, so keep this in mind when taking the photographs. Note that all images must be in jpg formate.

1. Place images in /usr/local/master/labs/rawphotos

2. Rename all images using the scheme [idnum]img.jpg where idnum is the id number of the piece of equipment photographed.

3. Run the conversion script

   ./convertImg.py

4. Enter angle to rotate photos. Photos from some cameras will look like they are properly orientated until they are posted on the website, at which time they will look like they are sideways. It is recommended that when using a new camera that the first image to add is used as a test case to determine if the images need to be rotated by the conversion script.

   Edited version will now appear in /usr/local/master/rawimages/output

5. Visually check all photographs in the output folder to confirm that they are still acceptable after they have been converted.

6. Move all photographs ready to be added to the database to /staffresources/equipment/equipimg

7. From /dev/python-tools run the script

   ./equipmentEdit.py −i

   to update the images in the equipmentDB.xml.

8. Check local version of website, and once the photos are acceptable remove all photos from /usr/local/-master/rawimages so that they are not added with the next batch of photos.

9. Update live version.

4.4   Adding Equipment Manuals

All manuals for equipment in inventory are contained in the folder /staffresources/equipment/equipman. Each piece of equipment can have as many manuals or other related documents as needed but they need to meet the following conditions.

- They must be in PDF format.

- They must be uniquely named.

- The name of must start with the equipment items four digit id number.

For example equipment item 0001 "Fluke Multimeter" can have manuals called 0001man.pdf, 0001diagram.pdf, 0001man-2.pdf.

Once all manuals to add have been added to the folder of manuals run...

./equipmentEdit.py −m

4.5 Deleting Old Equipment

NEED TO DEAL WITH MANUALS AND PHOTO OF DELETED EQUIPMENT

To remove a piece of old equipment run, from the python-tools folder, the command,

./equipmentEdit.py −d [idnum]

If the piece of equipment is currently listed as part of the equipment list for a current lab the script will prompt you to make sure that you know this. Ideally the equipment list in the lab repository should be updated first before removing the equipment. This will help to keep the lab equipment lists and the equipment database in sync.

5 Miscellaneous Content

5.1 Schedules

1. Create spreadsheet of schedules for the semester. (ex, schedule-WI2019.xlsx)

2. Create a PDF of the experiment schedule. (ex, schedule-WI2019.pdf)

3. Create a PDF of the lab room schedule. (ex, room-WI2019.pdf)

4. Place spreadsheet and PDFs in to folder /pjl-web/data/schedules

5. Copy /pjl-web/data/schedule-WI2019.pdf to /pjl-web/data/schedule-current.pdf

6. Copy /pjl-web/data/rooms-WI2019.pdf to /pjl-web/data/rooms-current.pdf

7. Add the previous schedule (ex, schedule-FA2018.pdf) to schedule archive

   nano /pjl−web/data/schedules/schedule−index.html

   Paste the code...

   ```
   <a href="/data/schedules/schedule−FA2018.pdf" target="_blank">
     <div class="schedule−download" >
       <i class="far fa−calendar" style="font−size: 20px;" aria−hidden=true></i>
       <div>
         <p>Experiment Schedule − Fall 2018</p> <!−− description −−>
       </div>
     </div>
   </a>
   ```

   ...directly before the similar code that exist for two semesters previous (ex, schedule-SU2018.pdf).

8. Add the previous schedule (ex, schedule-FA2018.pdf) to schedule archive

   nano /pjl−web/data/schedules/room−index.html

   Paste the code...

   ```
   <a href="/data/schedules/rooms−FA2018.pdf" target="_blank">
     <div class="schedule−download" >
       <i class="far fa−calendar" style="font−size: 20px;" aria−hidden=true></i>
       <div>
         <p>Rooms Schedule − Fall 2018</p> <!−− description −−>
       </div>
     </div>
   </a>
   ```

   ...directly before the similar code that exist for the two semesters ago.

5.2   Safety

5.2.1   Orientation

- Located in /pjl-web/data/safety/lab-safety-manual/
- Edit latest tex file and call it Orientation-WI2019.tex (adjusted for current semester and year)
- Overwrite file called Orientation.tex with updated version.
- Compile Orientation.tex
- **THERE IS A SYMLINK IN PARENT DIRECTORY. WHICH FILE DOES PJLDOCS LOOK FOR?**

5.3   Procedural Documents

NEED CONTENT

5.4   Templates

NEED CONTENT

6 Code

6.1 Scripts

6.1.1 Uploading Data to Live Website - liveUpdate.py

```python
#!/usr/bin/python3
#
# Script is to be run on web server to update contents of lab repository used in the live version
#
# Written by Peter Gimby, Nov 17 2017


import os, subprocess, argparse, filecmp, time
startTime = time.process_time()


'''define folder locations'''
root = "/usr/local/master/"
webSource = root + "pjl-web"
labSource = root + "labs"
webDest = "/mnt/local/pjl-web"
labDest = "/mnt/local/labs"
webMount = "/mnt/pjl-web-mnt"
labMount = "/mnt/lab-mnt"
devEquipXML = webSource + "/dev/equipmentDB.xml"
dataEquipXML = webSource + "/data/equipmentDB.xml"
liveEquipXML = webMount + "/data/equipmentDB.xml"

labFolders = ["downloads", "equipimg", "equipman", "landingpage", "repository", "safety", "schedules", "web-security"]
webFolders = ["css", "data", "dev", "doc", "fonts", "img", "js", "php", "repository", "staffresources"]
webFiles = ["index.html", "README.md"]
webFileReverse = ["equipmentDB.xml"]
mountInfo = [{"source": webSource, "mountPt": webMount}, {"source": labSource, "mountPt": labMount}]


'''define owners of files and general permissions'''
owner = "pgimby"
group = "pjl_admins"
apacheUser = "www-data"
devhost=["slug","fry"]
webserver="watt.pjl.ucalgary.ca"

def testHost(host):
    thishost = os.uname()[1]
    if thishost not in host:
        print("This script is designed to be run on " + thishost + " only.  Exiting...")
        gracefullExit(mountInfo)

def mountFolder(source,mountPoint,remote,option):
    fullSource = remote + ":" + source
```

```python
        os.system("mount -t nfs -o " + option + " " + fullSource + " " + mountPoint)
        if not os.system("mount | grep " + fullSource + " > /dev/null") == 0:
            print(fullSource + " did not mount properly. Exiting...")
            gracefullExit(mountInfo)


def umountFolder(mountPoint):
    os.system("umount " + mountPoint )


def syncFolder(testMode,source,dest):
    print("syning " + source)
    os.system("rsync" + testMode + " " + source + " " + dest)


def getDbFiles(dest,key):
    allFiles = os.listdir(dest)
    dbFiles = []
    for f in allFiles:
        if f.startswith(key) and f.split(".")[0][-1] in ['0','1','2','3','4','5','6','7']:
            dbFiles.append(f)
    return sorted(dbFiles)


def incrementFiles(files,dest,key,source,osTest):
    for i,f in enumerate(files):
        name = f.split(".")[0]
        index = int(name[-1])
        index += 1
        f = name[:-1] + str(index) + ".xml"
        os.system(osTest + "mv " + dest + "/" + files[i] + " " + dest + "/" + f)
    os.system(osTest + "mv " + dest + "/" + key + ".xml " + dest + "/" + key + "-0.xml")
    os.system(osTest + "cp " + source + "/" + key + ".xml " + dest + "/" + key + ".xml")


def wheel(dest,key,source,osTest):
    print("updating equipmentDB.xml")
    dbFiles = getDbFiles(dest,key)
    incrementFiles(list(reversed(dbFiles)),dest,key,source,osTest)


# def wheel(dbFile,source,dest,key,osTest):
# print("updating equipmentDB.xml")
# dbFiles = getDbFiles(dest,key)
# #incrementFiles(list(reversed(dbFiles)),dest,key,source,osTest)


def changePerm(varDir,owner,group,filePerm,options,osTest):
    processStart = time.process_time()

    print("changing permissions of " + varDir + " with find" + options + ". This may take a minute.")
    os.system(osTest + "find " + varDir + options + " -exec chmod " + filePerm + " {} \;")
    os.system(osTest + "find " + varDir + options + " -exec chown " + owner + "." + group + " {} \;")
    print("chargePerm Time: " + str(time.process_time() - processStart))


def gracefullExit(mountInfo):
    for i in mountInfo:
        umountFolder(i["mountPt"])
    exit()


'''checks that file a is newer that file b'''
def whichIsNewer(a,b,testMode):
    if os.path.isfile(a) and os.path.isfile(b):
        if os.path.getmtime(a) > os.path.getmtime(b):
            if testMode:
                print(a + " is newer than " + b )
                print(a + " " + str(os.path.getmtime(a)))
                print(b + " " + str(os.path.getmtime(b)))
            return True
        else:
            if testMode:
                print(b + " is newer than " + a)
                print(a + " " + str(os.path.getmtime(a)))
                print(b + " " + str(os.path.getmtime(b)))
            return False
```

```python
        else:
            if not os.path.isfile(a):
                print("File " + a + " Does not exist. Exiting...")
                gracefullExit(mountInfo)
            if not os.path.isfile(b):
                print("File " + b + " Does not exist. Exiting...")
                gracefullExit(mountInfo)
        # if os.path.getmtime(a) > os.path.getmtime(b):
        #     return True
        # else:
        #     return False


'''Main Script'''

'''User input to allow for a test mode during development'''
parser = argparse.ArgumentParser()
parser.add_argument('-t', '--test', help='test adding to xml without moving folders', action='store_true')
args = parser.parse_args()
testMode = args.test

if not os.getuid() == 0:
    print("This script must be run by \"The Great and Powerful Sudo\".")
    exit()

'''Parameters and options for operating in test mode'''
if testMode == True:
    rsycnOption = " -avnz --no-l"
    osTest = "echo "
else:
    rsycnOption = " -az --no-l"
    osTest = ""

'''Confirm that this script won't accidently run on the wrong machine'''
testHost(devhost)

'''mounts folder for syncing files and confirms success'''
mountFolder(webDest,webMount,webserver,"rw")
mountFolder(labDest,labMount,webserver,"rw")



'''update equipmenDB.xml from web server to development space if it is newer'''
if whichIsNewer(liveEquipXML,devEquipXML,testMode) and whichIsNewer(liveEquipXML,dataEquipXML,testMode):
    print("The live version of equipmentDB.xml is newer than the dev version.")
    if input("Do you wish to continue? y/N ") == "y":
        key = "equipmentDB"
        dataFolder = webSource + "/data"
        liveSource = webMount + "/data"
        wheel(dataFolder,key,liveSource,osTest)
        #wheel(i,source,dest,key,osTest)
    else:
        print("Exiting...")
        gracefullExit(mountInfo)


'''Set permissions and owners of files and folders'''
changePerm(labSource,owner,group,"644"," -type f",osTest)
changePerm(labSource,owner,group,"755"," -type d",osTest)
changePerm(webSource,owner,group,"644"," -type f",osTest)
changePerm(webSource,owner,group,"755", " -type d",osTest)

'''Sets the permission for executable'''
changePerm(webSource,owner,group,"750"," -type f -name \'*.py\'",osTest)

'''rsync lab content folders'''
for i in labFolders:
    source = labSource + "/" + i + "/"
    dest = labMount + "/" + i + "/"
    syncFolder(rsycnOption,source,dest)
```

```python
'''rsync webpage folders'''
for i in webFolders:
    source = webSource + "/" + i + "/"
    dest = webMount + "/" + i + "/"
    syncFolder(rsycnOption,source,dest)

'''rsync webpage files'''
for i in webFiles:
    source = webSource + "/" + i
    dest = webMount + "/"
    syncFolder(rsycnOption,source,dest)

'''changes the permissions of specific files and folders needed for live update of equipment numbers'''
changePerm(webMount + "/data" ,"root","www-data","660"," -type f -name equipmentDB.xml",osTest)
changePerm(webMount + "/data" ,"root","www-data","775"," -type d -name \'data\'",osTest)

'''replace .xml files in /dev folder with the ones in the /data folder'''
'''
Code still neeeded
'''

'''unmounts folders used for syncing files'''
umountFolder(webMount)
umountFolder(labMount)

print("Total Time: " + str(time.process_time() - startTime))

print("...and then there will be cake")
```

6.1.2 Modifying the Lab Document Database - repositoryEdit.py

```
#!/usr/bin/python3

'''
Can be called from the command line to make a wide range of changes to the lab repository xml file.
Change include

Adding versions of labs from a new semester
Adding a brand new lab to repository

color scheme
        red ERROR - FATAL
        green DEBUG
        cyan INPUT
        yellow WARNING

'''

'''Verison History

1.4 added the ability to customize the name give to a kit when only part of the kit is used.

'''
import pjlDB
import os, argparse, re, color
#import sys
#from termcolor import colored, cprint

version = "1.4.2"

#Fucntion that preform safety checks

def testHost(host):
        '''
        Test what computer this being run on. As of now it is machinne specific

        Args:
                host (str) name of host script was designed for

        Return:
                none
        '''
        thishost = os.uname()[1]
        if thishost not in host:
                print(color.red("[ERROR] ") + "This_script_is_designed_to_be_run_on_" + host + "_only._Exiting...")
                exit()


def checkTimeStamp(dev,data):
        '''
        Checks that the source files for the databases referenced are the latest. This protects against overwritting changes by
            ↪ mistake

        Args:
                dev (str) location of a file
                data (str) location of a file

        Return:
                (bool) True if file at data is newer than the one at dev
        '''
        if os.path.getmtime(data) <= os.path.getmtime(dev):
                return True
        else:
                return False


#Functions used to add a new version entry to the repositorsy xml
```

```python
'''creats a new empty lab object'''
def getLabObject(labdb):
        '''
        Used to generate a pjl lab object from the labDB.xml database

        Args:
                labdb (pjlDB.labDB) entire lab database object generated by pjlDB

        Return:
                lab (pjlDB._LabItem) individual lab item generated by pjlDB
        '''
        validID = False
        while not validID:
                idnum = input(color.grey("Enter_lab_ID_number:_"))
                if len(idnum) == 4 and idnum.isdigit() == True:
                        try:
                                lab = labdb.getLab(idnum)
                                validID = True
                        except pjlDB.IDDoesNotExist: ### not working properly
                                print("Message")
                else:
                        print(color.red("[ERROR]_") + "ID_formate_in_not_valid._Valid_IDs_are_of_the_form_####._
                        ↪ Please_try_again")
                        validID = False
        return lab


'''collects information about new version of an existing lab'''
def getVersionInfo(originalItem,validCourses,validSemesters,semesterKeys,eqdb,disciplineSource,topicSource,softwareSource,
    ↪ testMode):
        '''
        Main function that collects information for new version of lab entry

        Args:
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB
                validCourses (list) list of valid courses
                validSemesters (list) list of valid semesters
                semesterKeys (dict) dictionary that matches semesters with their abreviations
                eqdb (pjlDB.EquipDB) entire equipment inventory database object generated by pjlDB
                disiplineSource (str) path of file that contains all valid disciplines
                topicSource (str) path of file that contains all valid topics
                testMode (bool) allows script to be run in testing mode. No output written.

        Return:
                new_version (dict) dictionary that contains information needed for pjlDB package and a version of a lab to
                        ↪ an existing lab.
        '''
        print("Adding_version_to_\"" + originalItem.name + "\".")
        new_version = {}
        #new_version = {'originalDir': '/home/pgimby/labs/under−construction/211SP2018/lab2/', 'pdf': 'Phys 211_221 −
            ↪ Labatorial 02 − P2018.pdf', 'course': 'PHYS 211', 'semester': 'Spring', 'year': '2018', 'directory': '/data/
            ↪ repository/0003−Motion−on−the−Inclined−Plane/0003−PHYS211SP2018/', 'path': '/data/repository
            ↪ /0003−Motion−on−the−Inclined−Plane/0003−PHYS211SP2018/Phys 211_221 − Labatorial 02 − P2018.
            ↪ pdf'}
        new_version["idnum"] = originalItem.id_num
        new_version["name"] = originalItem.name
        new_version["type"] = originalItem.lab_type
        print("")
        new_version["originalDir"] = getOriginalDir()
        print("")
        new_version["pdf"] = getOriginalPdf(new_version["originalDir"])
        print("")
        new_version["course"] = validCourse(validCourses)
        print("")
        new_version["semester"] = validSemester(validSemesters)
        print("")
```

```python
        new_version["year"] = validYear()
        new_version["directory"],new_version["labFolder"] = validExistingDirectory(new_version,originalItem,semesterKeys)
        new_version["path"] = validPdfPath(new_version)
        new_version["equipment"] = getEquipList(eqdb,originalItem)
        print("")
        new_version["software"] = getSoftwareList(originalItem,softwareSource)
        print("")
        new_version['disciplines'] = getDisciplineList(originalItem,disciplineSource)
        print("")
        new_version['topics'] = getTopicList(originalItem,topicSource)
        return new_version

def getOriginalDir():
        '''
        Asks user for location of folder containing new lab, and check that it exists

        Args:
                none

        Return:
                originalDir (str) location of folder containing new lab
        '''
        validDir = False
        while not validDir:
                originalDir = input("Enter_absolute_path_for_directory_containing_lab:_")
                if not originalDir.split("/")[-1] == "":
                        originalDir = originalDir + "/"
                print(originalDir)
                if os.path.isdir(originalDir):
                        validDir = True
                else:
                        print("Directory_" + originalDir + "_does_not_exist._Please_try_again.")
        return originalDir

def getOriginalPdf(dir):
        '''
        Asks user for name of lab pdf file, and check that it exists

        Args:
                dir (str) pathname of the folder that the pdf should be in

        Return:
                pdfName (str) location of pdf file for new lab
        '''
        validPath = False
        while not validPath:
                pdfName = input("Enter_the_name_of_the_student_version_pdf:_")
                if os.path.isfile(dir + pdfName):
                        validPath = True
                else:
                        print("PDF_does_not_exist._Please_try_again.")
        return pdfName

def validCourse(validCourses):
        '''
        Asks user to enter the course the lab was used in, and checks it against a list of valid courses

        Args:
                validCourses (list) list of valid courses

        Return:
                course (str) valid course number
        '''
        validCourse = False
        while not validCourse:
```

```python
                courseNum = str(input("Enter course number: "))
                for i in validCourses:
                        if courseNum == i:
                                course = "PHYS " + courseNum
                                validCourse = True
                if not validCourse:
                        print("Invalid Course number")
                        print("Valid courses are...")
                        for i in validCourses:
                                print(i)
        return course

def validSemester(validSemesters):
        '''
        Asks user to enter the semester the lab was used in, and checks it against a list of valid semesters

        Args:
                validSemesters (list) list of valid courses

        Return:
                semesterName (str) valid semester name
        '''
        validSemester = False
        while not validSemester:
                semesterName = str(input("Enter semester: ")).capitalize()
                for i in validSemesters:
                        if semesterName == i:
                                validSemester = True
                if not validSemester:
                        print("Invalid semester")
                        print("Valid semesters are...")
                        for i in validSemesters:
                                print(i)
        return semesterName

def validYear():
        '''
        Asks user to enter the year the lab was used in, and checks that it is a valid year

        Args:
                none

        Return:
                year (str) valid 4 digit year
        '''
        validYear = False
        while not validYear:
                year = input("Enter year: ")
                if len(year) == 4 and year.isdigit() == True:
                        validYear = True
                else:
                        print("Year is invalid.")
        return year

def validExistingDirectory(new_version,lab,semesterKeys):
        '''
        Takes information entered from user, and determines the name of the folder that will
        contain version of lab that will be added. This uses knowledge of other version folder
        that have already been added. Will not work for a new Lab (see function versionFolder)

        Args:
                new_version (dict) information entered by user
                lab (pjlDB._LabItem) individual lab item generated by pjlDB
                semesterKeys (dict) matches abreviations for semesters with full name

        Return:
                directory (str) full path name on new version directory
        '''
```

```python
        samplePath = lab.versions[0]["directory"]
        labFolder = "/".join(samplePath.split("/")[:-1]) + "/"
        semester = semesterKeys[new_version["semester"]]
        courseNum = new_version["course"].split("_")[-1]
        year = new_version["year"]
        directory = labFolder + lab.id_num + "-PHYS" + courseNum + semester + year
        return directory,labFolder

def validPdfPath(new_version):
        '''
        Asks user to input the path to the pdf to display on the webpage for this version

        Args:
                new_version (dict) information entered by user

        Return:
                path (str) final path to pdf
        '''
        validPath = False
        while not validPath:
                pdfName = new_version["pdf"]
                if os.path.isfile(new_version["originalDir"] + pdfName):
                        validPath = True
                        path = new_version["directory"] + "/" + pdfName
                else:
                        print("PDF_does_not_exist._Please_try_again.")
        return path


def getEquipList(eqdb,originalItem):
        '''
        generates a list of equipment

        Args:
                eqdb (pjlDB.EquipDB) entire equipment inventory database object generated by pjlDB
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB

        Return:
                equipItems (list of dictionaries)
        '''
        print("")
        if input("Would_you_like_to_edit_the_equipment_list_for_this_lab?_y/N_").lower() == "y":
                print("")
                print("Current_Equipment_List")
                print("-------------------")
                for i in originalItem.equipment:
                        # i['alt-name'] = "TEST NAME"
                        # i['alt-id'] = "0000"
                        print(i['id'] + "_" + i['name'] + "_[" + i['alt-id'] + "_" + i['alt-name'] + "]_" + "_(" + i['amount'
                            ↪ ] + ")")
                equipItems = []
                equipItems = equipInfoReview(eqdb,originalItem)
                allItems = False
                while not allItems:
                        print("")
                        if input("Would_you_like_to_add_a_new_piece_of_equipment_for_this_lab?_y/N_").lower() == "y":
                                itemId = input("Enter_the_equipment_id_number:_")
                                equipItems.append(addEquipItem(eqdb,itemId))
                        else:
                                allItems = True
        else:
                equipItems = originalItem.equipment
        return equipItems

def equipInfoReview(eqdb,originalItem):
        '''
        Controls the review and editing of equipment list. Asks user to input id numbers and
        quantity of equipment needed for the new lab. User also can input an alternate/secondary
```

equipment item for each primary item

Input id numbers are check for correctness

Args:
  eqdb (pjlDB.EquipDB) entire equipment inventory database object generated by pjlDB
  originalItem (pjlDB._LabItem) individual lab item generated by pjlDB

Return:
  equipItems (list of dictionaries)
'''
```
        equipItems = []
        if originalItem.equipment:
                for i in originalItem.equipment:
                        print("ID_Number_[" + i['id'] + "]:_Name_[" + i['name'] + "]:_Alternate_Name:_[" + i['alt−name']
                            ↪ + "]:_Amount_[" + i['amount'] + "]" )
                        if input ("Would_you_like_to_edit_this_entry?_y/N:_").lower() == "y":
                                equipID = input("Enter_new_id_number_[" + i['id'] + "],_enter_'delete'_to_remove_this_
                                    ↪ item_")
                                if equipID == "":
                                        equipID = i['id']
                                if not equipID == "delete":
                                        item = addEquipItem(eqdb,equipID)
                                        print("enter_editing_code_here")
                                else:
                                        print("deleting_" + i['id'] + "_" + i['name'])
                        else:
                                equipItems.append(i)
        return equipItems

def addEquipItem(eqdb,itemId):
        '''
```
Adds a new piece of equipment to a lab object

Args:
  eqdb (pjlDB.EquipDB) entire equipment inventory database object generated by pjlDB
  itemId (str) equipment id number entered by user

Return:
  equipItem (dict) dicitonary for single equipment item
'''
```
        equipItem = {}
        validItem = False
        validAlt = False
        validNum = False
        itemName = ""
        altName = ""
        amount = ""

        # adds main item
        while not validItem:
                if itemId == "retry":
                        itemId = input("Enter_the_equipment_id_number:_")
                validItem,itemName,itemError = equipValid(eqdb,itemId)
                if not validItem:
                        print(itemError)
                        if input("Do_you_wish_to_try_again?_Y/n:_").lower() == "n":
                                break
                        else:
                                itemId = "retry"
                else:
                        equipItem['id'] = itemId
                        equipItem['name'] = itemName
                        validItem = True

        # adds alternate item
        while not validAlt:
                altId = input("Enter_id_number_of_an_alternate_for_this_item._If_none_hit_Enter._")
```

```python
                if not altId == "":
                        validAlt,altName,altError = equipValid(eqdb,altId)
                        if not validAlt:
                                print(altError)
                                if input("Do_you_wish_to_try_again?_Y/n:_").lower() == "n":
                                        break
                                else:
                                        altId = ""
                                        altName = ""
                                        validAlt = False
                        else:
                                equipItem['alt-name'] = altName
                                equipItem['alt-id'] = altId
                                validAlt = True
                else:
                        equipItem['alt-name'] = ""
                        equipItem['alt-id'] = ""
                        validAlt = True

        # adds the number of units needed
        while not validNum:
                amount = input("Please_enter_how_many_" + itemName + "(s)_are_needed?_")
                if amount.isdigit():
                        equipItem['amount'] = amount
                        validNum=True
                else:
                        print(amount + "_is_not_a_valid_number.")
                        if input("Do_you_wish_to_try_again?_Y/n:_").lower() == "n":
                                break
                equipItem["id"] = itemId
        return equipItem

def equipValid(eqdb,itemID):
        '''
        Checks if equipment item added by user for new lab is valid

        Args:
                eqdb (pjlDB.EquipDB) entire equipment inventory database object generated by pjlDB
                itemID (str) Id number of equipment item to add

        Return:
                validItem (bool)
                name (str) Name of equipment item
                errorMessage (str) Information on why a equipment entry is invalid
        '''
        errorMessage = ""
        if len(itemID) == 4 and itemID.isdigit() == True:
                try:
                        item = eqdb.getItem(idnum=itemID)
                        if item.is_kit:
                                name = equipNameOptions(item)
                        else:
                                name = item.name
                        errorMessage = ""
                        return True, name, errorMessage
                except pjlDB.EQIDDoesNotExist as e:
                        errorMessage = ("Invalid_Equipment:_Item_" + itemID + "_does_not_exist.")
                        name = "null"
                        return False, name, errorMessage
        else:
                errorMessage = ("Invalid_Equipment:_Id_needs_to_be_a_4_digit_number")
                name = "null"
                return False, name, errorMessage

def equipNameOptions(item):
        '''
        Sometimes a piece of a equipment is a piece of the kit. In these cases it can be useful for the name
        in the equipment list to appear as the part, but keep the equipment id number of the kit. This function
```

allows the user to choose a name from the list of components in a kit.

        Args:
                item (pjlDB.EquipDB) entire equipment inventory database object generated by pjlDB

        Return:
                name (str) Name of equipment item
        '''
        print()
        print(**type**(item))
        print("What_name_you_would_like_displayed_for_this_item?_Pick_a_number_[default_=_0].")
        print()
print("[0] " + item.name)

        kitItems = item.kit.split(",")
        kitItems[:0] = [item.name]
        for i **in range**(0, **len**(kitItems)):
                index = **str**(i)
                print("[" + index + "]_" + kitItems[i].strip())
        validName = False
        while not validName:
                selection = **int**(**input**("Please_Choose_a_Name_for_This_Item?_[0]_"))
                #if 1 <= selection <= len(kitItems) + 1:
                if 0 <= selection <= **len**(kitItems):
                        name = kitItems[selection].strip()
                        validName = True
                elif selection == 0:
                        name = item.name
                        validName = True
                else:
                        print(selection + "_is_not_a_option.")
                        if input("Do_you_wish_to_try_again?_Y/n:_").lower() == "n":
                                exit()
        return name
                #print(kitItems)
        # i =+ 1
        #print(item.is_kit)
        #print(item.kit)

        # print()
        # print(type(item))
        # print("What name you would like displayed for this item? Pick a number [default = 0].")
        # print()
        # print("[0] " + item.name)

        # kitItems = item.kit.split(",")
        # #kitItems.insert(item.name)
        # for i in range(len(kitItems)):
        # index = str(i + 1)
        # print("[" + index + "] " + kitItems[i].strip())
        # validName = False
        # while not validName:
        # selection = int(input("Please Choose a Name for This Item? [0] "))
        # #if 1 <= selection <= len(kitItems) + 1:
        # if 0 <= selection <= len(kitItems):
        #   name = kitItems[selection].strip()
        #   validName = True
        # elif selection == 0:
        #   name = item.name
        #   validName = True
        # else:
        #   print(selection + " is not a option.")
        #   if input("Do you wish to try again? Y/n: ").lower() == "n":
        #   exit()
        # return name
        # #print(kitItems)
        # # i =+ 1

```python
        # #print(item.is_kit)
        # #print(item.kit)


def getSoftwareList(originalItem,softwareSource):
        '''
        generates a list of software

        Args:
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB
                softwareSource (string) path to file containing list of available software

        Return:
                softwareItems (list) valid software
        '''
        softwareItems = []
        if input("Would you like to edit the software list for this lab? y/N ").lower() == "y":
                print("")
                print("Current Software")
                print("-------------------")
                for i in originalItem.software:
                        print(i)
                print("")
                softwareItems = []
                softwareItems = softwareRemove(originalItem)
                allItems = False
                while not allItems:
                        if input("Would you like to add a new software for this lab? y/N ").lower() == "y":
                                print("")
                                masterList = getValidList(softwareSource)
                                print("")
                                print("Valid Software")
                                printList(masterList)
                                softwareItems.append(getNewSoftware(masterList))
                        else:
                                allItems = True
                        softwareItems = list(set(softwareItems))
        else:
                softwareItems = originalItem.software
        return softwareItems

def softwareRemove(originalItem):
        '''
        Removes unwanted software

        Args:
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB

        Return:
                softwareItems (list) wanted software
        '''
        softwareItems = []
        if originalItem.software:
                for i in originalItem.software:
                        if not input("Would you like to remove \"" + i + "\" as needed software? If so enter 'delete': ")
                        ↪ .lower() == "delete":
                                softwareItems.append(i)
        return softwareItems

def getNewSoftware(masterList):
        '''
        Get list of software from user and check if they are valid

        Args:
                masterList (list) complete pool of valid topics

        Return:
```

```
                    software (str) single valid software for new lab
        '''
        valid = False
        while not valid:
                item = input("Enter_new_software:_")
                for i in masterList:
                        if i.lower() == item.lower():
                                valid = True
                                item = i
                                print("Adding_" + i + "_to_software")
                                print("")
                if not valid:
                        print(item + "_is_invalid_software.")
                        if not input("Would_you_like_to_try_again?_Y/n_").lower() == "n":
                                continue
                        else:
                                break
        return item

def getValidList(listSource):
        '''
        Generates list of valid selections from file

        Args:
                listSource (string) path to file containing valid list entries

        Return:
                validList (list) list of valid selections

        '''
        validList = open(listSource).readlines()
        for i in range (0,len(validList)):
                validList[i] = validList[i].replace('\n','').strip()
        validList = list(filter(None, validList))
        return validList

def getDisciplineList(originalItem,disciplineSource):
        '''
        generates a list of disciplines

        Args:
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB
                disciplineSource (string) path to file containing list of disciplines

        Return:
                disciplineItems (list) valid disciplines
        '''
        disciplineItems = []
        if input("Would_you_like_to_edit_the_disciplines_list_for_this_lab?_y/N_").lower() == "y":
                print("")
                print("Current_Disciplines")
                print("-------------------")
                for i in originalItem.disciplines:
                        print(i)
                print("")
                disciplineItems = []
                disciplineItems = disciplineRemove(originalItem)
                allItems = False
                while not allItems:
                        if input("Would_you_like_to_add_a_new_discipline_for_this_lab?_y/N_").lower() == "y":
                                print("")
                                masterList = getValidList(disciplineSource)
                                print("")
                                print("Valid_Disciplines")
                                printList(masterList)
                                disciplineItems.append(getNewDisciplines(masterList))
                        else:
                                allItems = True
```

```python
                disciplineItems = list(set(disciplineItems))
        else:
                disciplineItems = originalItem.disciplines
        return disciplineItems

def disciplineRemove(originalItem):
        '''
        Removes unwanted disciplines

        Args:
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB

        Return:
                disciplineItems (list) wanted disciplines
        '''
        disciplineItems = []
        if originalItem.disciplines:
                for i in originalItem.disciplines:
                        if not input("Would_you_like_to_remove_\"" + i + "\"_as_a_discipline?_If_so_enter_'delete':_").
                        ↪ lower() == "delete":
                                disciplineItems.append(i)
        return disciplineItems

def getNewDisciplines(masterList):
        '''
        Get list of disciplines from user and check if they are valid

        Args:
                masterList (list) complete pool of valid topics

        Return:
                disciplines (str) single valid discipline for new lab
        '''
        valid = False
        while not valid:
                item = input("Enter_new_discipline:_")
                for i in masterList:
                        if i.lower() == item.lower():
                                valid = True
                                item = i
                                print("Adding_" + i + "_to_disciplines")
                                print("")
                if not valid:
                        print(item + "_is_an_invalid_discipline.")
                        if not input("Would_you_like_to_try_again?_Y/n_").lower() == "n":
                                continue
                        else:
                                break
        return item

def getTopicList(originalItem,topicSource):
        '''
        generates a list of topics

        Args:
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB
                topicSource (string) path to file containing list of topics

        Return:
                topicItems (list) valid topics
        '''
        topicItems = []
        if input("Would_you_like_to_edit_the_topics_list_for_this_lab?_y/N_").lower() == "y":
                print("")
                print("Current_Topics")
                print("------------------")
                for i in originalItem.topics:
                        print(i)
```

```python
                        print("")
                        topicItems = []
                        topicItems = topicRemove(originalItem)
                        print("")
                        allItems = False
                        while not allItems:
                                if input("Would_you_like_to_add_a_new_topic_for_this_lab?_y/N_").lower() == "y":
                                        print("")
                                        masterList = getValidList(topicSource)
                                        print("")
                                        print("Valid_Topics")
                                        printList(masterList)
                                        topicItems.append(getNewTopic(topicSource,masterList))
                                else:
                                        allItems = True
                        topicItems = list(set(topicItems))
                else:
                        topicItems = originalItem.topics
        return topicItems

def topicRemove(originalItem):
        '''
        Removes unwanted topics

        Args:
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB

        Return:
                topicItems (list) wanted topics
        '''
        topicItems = []
        if originalItem.topics:
                for i in originalItem.topics:
                        if not input("Would_you_like_to_remove_\"" + i + "\"_as_a_topic?_If_so_enter_'delete':_").lower()
                            ↪ == "delete":
                                topicItems.append(i)
        return topicItems

def getNewTopic(topicSource,masterList):
        '''
        Get list of topics from user and check if they are valid

        Args:
                topicSource (string) path to file containing list of disciplines
                masterList (list) complete pool of valid topics

        Return:
                topics (str) single valid topic
        '''
        valid = False
        while not valid:
                item = input("Enter_new_topic:_")
                for i in masterList:
                        if i.replace("","").lower() == item.replace("","").lower():
                                valid = True
                                item = i
                                print(item)
                                print("Adding_" + i + "_to_topics")
                                print("")
                if not valid:
                        print(item + "_is_an_invalid_topic.")
                        if not input("Would_you_like_to_try_again?_Y/n_").lower() == "n":
                                continue
                        else:
                                break
        return item

def printList(lst):
```

```python
        '''
        Prints a list of strings line by line for easy readability

        Args:
                lst (list) list of strings to be printed

        Return:
                none
        '''
        print("------------------")
        for i in lst:
                print(i)
        print("")

def confirmEntry(new_version):
        '''
        print out what information entered by usee, and asks for confirmation

        Args:
                new_version (dict) dictionary containing all data in for that pjlDB can enter into database

        Return:
                (bool) True if information has been confirmed by user
        '''
        print("")

        print("Please confirm that the information entered is correct")
        print("lab id: " + new_version["idnum"])
        print("name: " + new_version["name"])
        print("type: " + new_version["type"])
        print("original Directory: " + new_version["originalDir"])
        print("course: " + new_version["course"])
        print("semester: " + new_version["semester"])
        print("year: " + new_version["year"])
        print("directory: " + new_version["directory"])
        print("path: " + new_version["path"])
        print("equipment: ")
        printList(new_version["equipment"])
        print("disciplines: ")
        printList(new_version["disciplines"])
        print("topics: ")
        printList(new_version["topics"])
        if not input("Is this information correct? N/y: ").lower() == "y":
                print("exiting...")
                exit()

def validDB(info,lab,labdb):
        '''
        adds lab object to database and checks that database is valid

        Args:
                info (dict) information about new lab object
                lab (pjlDB._LabItem) individual lab item generated by pjlDB
                labdb (pjlDB.LabDB) entire lab database object generated by pjlDB

        Return:
                (bool) True if labDB object is valid
        '''
        lab.id_num = info["idnum"]
        lab.name = info["name"]
        lab.lab_type = info["type"]
        lab.equipment = info["equipment"]
        lab.software = info["software"]
        lab.disciplines = info["disciplines"]
        lab.topics = info["topics"]
        lab.addVersion(info)
        valid = labdb.validateFull()
        if valid:
```

```python
                return valid
        else:
                return False


#Functions for moving directory into repository

def validDir(info,root):
        '''
        checks that the verison has not already been added to repository file structure

        Args:
                info (dict) information about new lab object
                root (str) root path of lab repository

        Return:
                (bool) True is lab has not already been added to repository file structure
        '''
        versionDir = root + info["directory"]
        if not os.path.isdir(versionDir):
                return True
        else:
                print("Lab folder " + versionDir + " Already Exists.")
                print("Exiting...")
                return False

def moveVersionDir(info,root):
        '''
        adds source file to lab repository.
                Makes new directory
                rsyncs files except for contents of Support_Docs folder

        Args:
                info (dict) information about new lab object
                root (str) root path of lab repository

        Return:
                none
        '''
        versionDir = root + info["directory"]
        if not os.path.isdir(versionDir):
                os.system("mkdir " + versionDir)
                #os.system("echo rsync -avz --exclude Support_Docs " + info["originalDir"] + " " + versionDir)
                os.system("sudo rsync -avz --exclude Support_Docs " + info["originalDir"] + " " + versionDir)
        else:
                print("")
                print("Lab folder " + versionDir + " Already Exists.")
                print("")
                if input("Do you want to update the folder contents? N/y: ").lower() == "y":
                        print("")
                        os.system("rsync -avz --exclude Support_Docs " + info["originalDir"] + " " + versionDir)
                else:
                        print("Exiting...")
                        exit()

#Functions for updating Support_Docs

def addSupportFolder(info,root):
        '''
        adds contents of Support_Docs folder to repository

        Args:
                info (dict) information about new lab object

        Return:
                none
        '''
        originDir = info["originalDir"] + "Support_Docs"
```

```python
            destinationDir = root + info["labFolder"] + "/Support_Docs"
        if os.path.isdir(originDir):
                if not os.path.isdir(destinationDir):
                        print("Support_Docs_Folder_does_not_exist._Adding_new_folder_" + destinationDir)
                        os.system("mkdir_" + destinationDir)
                if os.path.isdir(destinationDir):
                        os.system("rsync_-avz_" + originDir + "/_" + destinationDir)
                else:
                        print("Something_when_wrong._Exiting...")
                        exit()


# Functions for adding a new lab

def getNewLabInfo(originalItem,testMode):
        '''
        Main function that collects information for new lab entry

        Args:
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB
                validCourses (list) list of valid courses
                validSemesters (list) list of valid semesters
                semesterKeys (dict) dictionary that matches semesters with their abreviations
                eqdb (pjlDB.EquipDB) entire equipment inventory database object generated by pjlDB
                disiplineSource (str) path of file that contains all valid disciplines
                topicSource (str) path of file that contains all valid topics
                testMode (bool) allows script to be run in testing mode. No output written.

        Return:
                new_lab (dict) dictionary that contains information needed for pjlDB package to create new lab object
                labFolder (str) path of parent folder to create for new lab
        '''
        new_lab = {}
        new_versions = []
        new_version = {}
        new_lab["idnum"] = originalItem.id_num
        print("Adding_new_lab_with_id_number:_[" + new_lab["idnum"] + "]")
        print("-----------------------------------")
        print("")
        new_lab["name"] = getName(originalItem)
        print("")
        new_lab["type"] = getType()
        print("")
        new_lab["originalDir"] = getOriginalDir()
        print("")
        new_lab["pdf"] = getOriginalPdf(new_lab["originalDir"])
        print("")
        new_lab["course"] = validCourse(validCourses)
        print("")
        new_lab["semester"] = validSemester(validSemesters)
        print("")
        new_lab["year"] = validYear()
        print("")
        new_version
        new_lab["labFolder"],new_lab["labFolderPath"] = newLabFolder(new_lab)
        print("----------------------")
        print(new_lab["labFolderPath"])
        print(new_lab["labFolder"])
        new_lab["directory"] = versionFolder(new_lab,semesterKeys)
        print(new_lab["directory"])
        new_lab["path"] = validPdfPath(new_lab)
        # # Start tmp code
        # new_lab = {'idnum': '0189', 'name': 'NEW-NAME', 'type': 'Lab', 'originalDir': '/home/pgimby/labs/under-
            ↪ construction/365/ready/tutorial-lab/', 'pdf': 'Interference-Numerical-F2019.pdf', 'course': 'PHYS 365', '
            ↪ semester': 'Fall', 'year': '2019', 'labFolder': '/data/repository/0189-NEW-NAME', 'labFolderPath': '/usr/
            ↪ local/master/pjl-web/data/repository/0189-NEW-NAME', 'directory': '/data/repository/0189-NEW-
            ↪ NAME/0189-PHYS365FA2019', 'path': '/data/repository/0189-NEW-NAME/0189-PHYS365FA2019/
            ↪ Interference-Numerical-F2019.pdf'}
```

```python
        # # End Tmp code
        new_lab["equipment"] = getEquipList(eqdb,originalItem)
        print("")
        new_lab["software"] = getSoftwareList(originalItem,softwareSource)
        print("")
        new_lab['disciplines'] = getDisciplineList(originalItem,disciplineSource)
        print("")
        new_lab['topics'] = getTopicList(originalItem,topicSource)
        new_version["path"] = new_lab["path"]
        new_version["year"] = new_lab["year"]
        new_version["semester"] = new_lab["semester"]
        new_version["course"] = new_lab["course"]
        new_version["directory"] = new_lab["directory"]
        new_versions.append(new_version)
        new_lab["versions"] = new_versions
        return new_lab

def getName(originalItem):
        '''
        Asks user to enter name of new lab, and check that the name is not already used

        Args:
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB

        Return:
                labName (str) name of new lab
        '''
        labName = originalItem.name
        if originalItem.name == "":
                validName = False
                while not validName:
                        labName = str(input("Enter_name_of_the_new_lab._Please_use_conventional_titlecase_(ie._This_is_
                            ↪ a_Title_of_a_New_Lab):_"))
                        if input("Is_this_name_entered_correctly?_N/y:_").lower() == "y":
                                validName = True
                        elif input("Would_you_like_to_try_again?_Y/n_").lower() == "n":
                                print("Exiting.")
                                exit()
        return labName

def getType():
        '''
        asks user what type of experiment this is. There are only two options lab or labatorial

        Args:
                none

        Return (str) type experiment. lab or labatorial
        '''
        validType = False
        while not validType:
                labType = input("Is_this_a_lab_or_a_labatorial?_").lower()
                if labType == "lab" or labType == "labatorial":
                        validType = True
                else:
                        if input("Would_you_like_to_try_again?_Y/n:_").lower() == "n":
                                exit()
        return labType.capitalize()

def newLabFolder(info):
        '''
        determine name of folder for a new lab

        Args:
                info (dict) information about new lab object
        '''
        name = "-".join(info["name"].split("_"))
        labFolder = "/data/repository/" + info["idnum"] + "-" + name
```

```python
        labFolderPath = root + labFolder
        return labFolder,labFolderPath


def versionFolder(info,semesterKeys):
        '''
        determine name of folder for a new lab. This is different than the function
        (validExistingDirectory) which uses knowledge of existing version folders.

        Args:
                info (dict) information about new lab object
                labFolder (str) path of the new lab parent folder in repository
                semesterKeys (dict) matches full name semesters with abreviation

        Return:
                directory (str) path of version directory for a new lab
        '''
        semester = semesterKeys[info["semester"]]
        courseNum = info["course"].split("_")[-1]
        directory = info["labFolder"] + "/" + info["idnum"] + "-PHYS" + courseNum + semester + info["year"]
        return directory


def validNewLab(info,lab,labdb):
        '''
        adds lab object to database and checks that database is valid

        Args:
                info (dict) information about new lab object
                lab (pjlDB._LabItem) individual lab item generated by pjlDB
                labdb (pjlDB.LabDB) entire lab database object generated by pjlDB

        Return:
                (bool) True if labDB object is valid
        '''
        lab.id_num = info["idnum"]
        lab.name = info["name"]
        lab.lab_type = info["type"]
        lab.equipment = info["equipment"]
        lab.software = info["software"]
        lab.disciplines = info["disciplines"]
        lab.topics = info["topics"]
        lab.versions = info["versions"]
        labdb.addLab(lab)
        valid = labdb.validateFull()
        if valid:
                return valid
        else:
                return False

# Functions for editing existing xml entries

def getEditInfo(originalItem,eqdb,disciplineSource,topicSource,softwareSource,testMode):
        '''
        Main function that collects information for new version of lab entry

        Args:
                originalItem (pjlDB._LabItem) individual lab item generated by pjlDB
                eqdb (pjlDB.EquipDB) entire equipment inventory database object generated by pjlDB
                disiplineSource (str) path of file that contains all valid disciplines
                topicSource (str) path of file that contains all valid topics
                testMode (bool) allows script to be run in testing mode. No output written.

        Return:
                originalItem (pjlDB._LabItem) updated individual lab item
                updatedVersion (dict) new information related to an updated version
        '''
        print("Editing_info_for_\"" + originalItem.name + "\".")
        originalItem.equipment = getEquipList(eqdb,originalItem)
        print("")
```

```python
        originalItem.software = getSoftwareList(originalItem,softwareSource)
        print("")
        originalItem.disciplines = getDisciplineList(originalItem,disciplineSource)
        print("")
        originalItem.topics = getTopicList(originalItem,topicSource)
        print("")
        updatedVersion = {}
        if input("Would you like to updated a version of  lab? y/N ").lower() == "y":
#   if not testMode:
            updatedVersion = updateVersionInfo(originalItem)
            originalItem.versions = updateVersion(originalItem,updatedVersion)
#   else:
#       updatedVersion = {'course': 'PHYS 227', 'semester': 'Fall', 'year': '2019', 'originalDir': '/home/pgimby/labs/under-
#       ↪ construction/227FA2019/0023/', 'pdf': 'Lab-01-IntroToComputationalModelling.pdf', 'directory': '/data/repository
#       ↪ /0023-Introduction-to-3D-Computational-Modelling/0023-PHYS227FA2019', 'labFolder': '/data/repository
#       ↪ /0023-Introduction-to-3D-Computational-Modelling/', 'path': '/data/repository/0023-Introduction-to-3D-
#       ↪ Computational-Modelling/0023-PHYS227FA2019/Lab-01-IntroToComputationalModelling.pdf'}
#       tmp = updateVersion(originalItem,updatedVersion)
#       print("")
                        #moveVersionDir(updatedVersion,root)
                        #tmp2 = validDB(updatedVersion,originalItem,labdb)
                        #print(tmp2)
        return originalItem, updatedVersion


def displayLabItem(lab):
        print("")
        print("Please confirm that the information entered is correct.")
        print("------------------------------------------------------------------")
        print("lab id: " + lab.id_num)
        print("")
        print("name: " + lab.name)
        print("")
        print("type: " + lab.lab_type)
        print("")
        print("equipment: ")
        print("----------")
        printEquipList(lab.equipment)
        print("")
        print("disciplines: ")
        printList(lab.disciplines)
        print("topics: ")
        printList(lab.topics)
        if input("Is this information correct? N/y: ").lower() == "y":
                return True
        else:
                print("exiting...")
                return
                exit()


def printEquipList(equipList):
        for i in equipList:
                print(i["id"] +": " + i["name"] + " (" + i["amount"] + "), " + i["alt-id"] + ": " + i["alt-name"])


# Functions for updating an existing version of a lab.

def getDeploymentID(version):
        '''
        Takes a dictionaries containing version of a lab, and pulls out a unique
        identifier based on when a version was deployed, and in what course.
        The result is in the form 227FA2019.

        Args:
                version (dict)

        Return:
                deploymentId (string) unique identifer for when and where lab was used.
```

```python
        '''
        course = version["course"][5:]
        year = version["year"]
        semester = version["semester"]
        key = semesterKeys[semester]
        deploymentId = course + key + year
        return deploymentId


def updateVersionInfo(originalItem):
        '''
        Gathers information from the user as to what file to update, and where to
        find the new version

        Args:
                originalItem (pjlDB._LabItem) original lab object

        Return:
                updatedVersion (dict) dictionary containing information about updated
                lab version.
        '''
        print("")
        updatedVersion = {}
        updatedVersion["course"] = validCourse(validCourses)
        print("")
        updatedVersion["semester"] = validSemester(validSemesters)
        print("")
        updatedVersion["year"] = validYear()
        print("")
        updatedVersion["originalDir"] = getOriginalDir()
        print("")
        updatedVersion["pdf"] = getOriginalPdf(updatedVersion["originalDir"])
        print(updatedVersion["pdf"])
        updatedVersion["directory"],updatedVersion["labFolder"] = validExistingDirectory(updatedVersion,originalItem,
            ↪ semesterKeys)
        print("")
        updatedVersion["path"] = validPdfPath(updatedVersion)
        return updatedVersion



def updateVersion(originalItem,updatedVersionInfo):
        '''
        Takes a dictionaries containing versions of a lab, and replaces the pdf
        with the a new version as per user input

        Args:
                originalItem (pjlDB._LabItem) original lab object
                updatedVersionInfo (dict) updated information on the lab version

        Return:
                lab.versions (list) of dictionaries, one entry for each version.
        '''
        newDeployId = getDeploymentID(updatedVersionInfo)
        for i in lab.versions:
                if getDeploymentID(i) == newDeployId:
                        i["path"] = updatedVersionInfo["path"]
        return lab.versions

#Main Script


'''List of valid courses and semesters'''
validCourses = ["211", "223", "227", "255", "259", "323", "325", "341", "365", "369", "375", "397", "497"]
validSemesters = ["Winter", "Spring", "Summer", "Fall"]
semesterKeys = {"Winter": "WI", "Spring": "SP", "Summer": "SU", "Fall": "FA"}


'''Define user options'''
parser = argparse.ArgumentParser(
```

```python
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog='''
Know bugs and other important information:
-----------------
        A) Blank Topics or Discplies was causing issures. I think
        B) Spaces in the names of folders does not work.
        C) Review does not include software.
        D) Cannot add additional docs.
''')
parser.add_argument('-a', '--add', help='Add_a_new_version_to_an_existing_lab.".', action='store_true')
parser.add_argument('-e', '--edit', help='Edit_the_details_of_a_lab.', action='store_true')
parser.add_argument('-n', '--new', help='Add_a_brand_new_lab.".', action='store_true')
parser.add_argument('-t', '--test', help='Debug_mode.', action='store_true')
parser.add_argument('-x', '--validate', help='Disable_validation_for_xml.', action='store_true')
parser.add_argument('-v', '--version', help='Print_current_verion_of_script.', action='store_true')
args = parser.parse_args()
testMode = args.test
validate = args.validate


'''Paths for files'''
root = "/usr/local/master/pjl-web"
eqdbDev = root + "/dev/equipmentDB.xml"
labdbDev = root + "/dev/labDB.xml"
eqdbData = root + "/data/equipmentDB.xml"
labdbData = root + "/data/labDB.xml"
disciplineSource = root + "/data/validDisciplines.txt"
topicSource = root + "/data/validTopics.txt"
softwareSource = root + "/data/validSoftware.txt"



'''Changes the output to a temporary file if script is run in test mode'''
if testMode:
        destXML = root + "/dev/test_labDB.xml"
        print("----------Running_in_test_mode.----------")
else:
        destXML= labdbDev

'''validation disabled warning'''
if validate:
        print("validation_of_output_file_has_been_disabled._Be_Very_Careful!")

'''name of host machine this scipt was written for'''
#devhost = "slug"
devhost = ["slug","fry"]



'''Confirm that this script won't accidently run on the wrong machine'''
testHost(devhost)



'''Create pjlDB object of each of the relevent xml files'''
eqdb = pjlDB.EquipDB(eqdbDev)
labdb = pjlDB.LabDB(labdbDev)



'''prints version'''
if args.version:
        print("Version_" + version)
        exit()



'''Checks that the development version of both key DBs are new or as new as the live versions.'''
if not checkTimeStamp(eqdbDev,eqdbData) or not checkTimeStamp(labdbDev,labdbData):
        if not checkTimeStamp(eqdbDev,eqdbData):
                print("Equipment_development_database_is_out_of_synce_with_the_live_version._Please_update_the_
                ↪ development_version_before_continuing.")
        if not checkTimeStamp(labdbDev,labdbData):
                print("Repository_development_database_is_out_of_synce_with_the_live_version._Please_update_the_
```

```
                                ↪ development_version_before_continuing.")
        print("Exiting...")
        exit()


'''add a new version of an existing lab'''
if args.add:
        print("Adding_new_lab_version.")
        lab = getLabObject(labdb)
        versionInfo = getVersionInfo(lab,validCourses,validSemesters,semesterKeys,eqdb,disciplineSource,topicSource,
                ↪ softwareSource,testMode)
        confirmEntry(versionInfo)
        if validDB(versionInfo,lab,labdb) and validDir(versionInfo,root):
                labdb.save(destXML, ignore_validation=validate, error_log=True)
                if not testMode:
                        moveVersionDir(versionInfo,root)
                        addSupportFolder(versionInfo,root)
        else:
                print("something_when_wrong")
                exit()

'''create a new lab'''
if args.new:
        print("Adding_new_lab.")
        lab = labdb.newLab(labdb.new_id)
        newLabInfo = getNewLabInfo(lab,testMode)
        lab.name = newLabInfo["name"]
        lab.type = newLabInfo["type"]
        confirmEntry(newLabInfo)
        if validDB(newLabInfo,lab,labdb):
                if not testMode:
                        os.system("mkdir_" + newLabInfo["labFolderPath"])
                        moveVersionDir(newLabInfo,root)
                        addSupportFolder(newLabInfo,root)
                        labdb.addLab(lab)
                        labdb.save(destXML, ignore_validation=validate, error_log=True)
                else:
                        os.system("echo_mkdir_" + newLabInfo["labFolder"])
                        labdb.addLab(lab)
                        labdb.save(destXML, ignore_validation=validate, error_log=True)


if args.edit:
        print("Editing_existing_lab.")
        lab = getLabObject(labdb)
        lab, versionInfo = getEditInfo(lab,eqdb,disciplineSource,topicSource,softwareSource,testMode)
        if displayLabItem(lab):
                labdb.save(destXML, ignore_validation=validate, error_log=True)
                if bool(versionInfo):
                        moveVersionDir(versionInfo,root)

'''confirms that the script has ended properly'''
print("...and_then_there_will_be_cake")
```

6.1.3 Modifying the Inventory Database - equipmentEdit.py

#!/usr/bin/python3

'''This script is designed to add edit the equipment inventory. It can be used to add new equipment or to remove an existing
    ↪ *piece of equipment'''*

 # Version 1.1 added image update function, manual update function
 # Version 1.2 added ability to edit equipment information of existing items

#Future Version
#Add logging system to track changes

import pjlDB
import os, argparse, unicodedata, re
import xml.etree.ElementTree as ET

#Fucntion that preform safety checks

def testHost(host):
    '''
      Test what computer this being run on. As of now it is machinne specific

      Args:
            host (str) name of host script was designed for

      Return:
            none
    '''
    thishost = os.uname()[1]
    if thishost **not in** host:
        print("This script is designed to be run on " + thishost + " only. Exiting...")
        exit()

def checkTimeStamp(dev,data):
    '''
      Checks that the source files for the databases referenced are the latest. This protects against overwritting changes by
        ↪ *mistake*

      Args:
            dev (str) location of a file
            data (str) location of a file

      Return:
            (bool) True if file at data is newer than the one at dev
    '''
    if os.path.getmtime(data) <= os.path.getmtime(dev):
        return True
    else:
        return False

 # Function used for deleting equipment

'''Functions used for deleting equipiment items'''
def deleteEquipItem(eqdb,labdb):
    valid = False
    while not valid:
        itemID = **input**("Please enter the id number of the equipment item you wish to remove? ")
        if len(itemID) == 4 **and** itemID.isdigit() == True:
            try:
                item = eqdb.getItem(idnum=itemID)
                if input("Do you want to delete – " + item.name + "? (y/N) ") == "y":
                    eqdb.deleteItem(labdb, itemID)
            except pjlDB.EQIDDoesNotExist as e:

```python
                        print(e)
                    valid = True
                else:
                    print("Invalid equipment id number entered. Needs to be a 4 digit number")
                    return False


#Functions for collecting information for a new or existing equipment item

'''Gets an existing equipment item from xml'''
def getItemToEdit(eqdb):
        valid = False
        while not valid:
                eqID = input("Enter the id number of the equipment item you wish to edit. ")
                if len(eqID) == 4 and eqID.isdigit() == True:
                        try:
                                eqItem = eqdb.getItem(idnum=eqID)
                                valid = True
                                return eqItem
                        except pjlDB.EQIDDoesNotExist as e:
                                print(e)
                                if input("Would you like to try again? (y/N)") == "y":
                                        valid = False
                                else:
                                        exit()
                else:
                        print(eqID + " is not a valid equipment id number. Must be a four digit number.")

'''Collects information of kit status and/or contents'''
def getKit(oldItem):
        kit = []
        kitString = ""
        name = ""
        print(oldItem.kit)
        kitStatus = input("Is this item a kit? (y/N) [" + str(oldItem.is_kit) + "] ").lower()
        if kitStatus == "y":
                iskit = True
        elif kitStatus == "n":
                iskit = False
        else:
                iskit = oldItem.is_kit
        if iskit:
                name = input("Name of kit: [" + oldItem.name + "]")
                if name == "":
                        name = oldItem.name
                if input("Would you like to edit the kit contents? y/N ").lower() == "y":
                        lastItem = False
                        kit = editKitList(oldItem)
                        if not input("Would you like to add another item? Y/n ").lower() == "n":
                                kit = addNewKitItem(kit)
                        kitString = ", ".join(kit )
                else:
                        kitString = oldItem.kit
        return iskit, name, kitString

def editKitList(oldItem):
        kit = []
        origKitList = oldItem.kit.split(",")
        for i in origKitList:
                name = i.split("(")[0].strip()
                try:
                        amount = i.split("(")[1].strip()
                        amount = str(amount.split(")")[0])
                except:
                        amount = str(1)
                newName = input("Enter new name [" + name + "] or enter \'delete\' to remove ")
                if newName == "delete":
                        continue
```

```python
                elif not newName == "":
                        name = newName
                print(name)
                newAmount = input("Enter amount of " + name + "(s) [" + amount + "]: ")
                if newAmount.isdigit():
                        amount = str(newAmount)
                if amount == "1":
                        item = name
                else:
                        item = name + " (" + amount +")"
                kit.append(item)
        return kit

def addNewKitItem(kit):
        lastItem = False
        while not lastItem:
                kitItemName = input("Kit Item : ")
                if kitItemName == "":
                        print("Name is blank. Please try again, or enter \'pass\' to continue.")
                        continue
                if kitItemName.lower() == "pass":
                        lastItem = True
                        continue
                kitItemAmount = input("How many " + kitItemName + "(s) are there? ")
                if kitItemAmount == "1":
                        kit.append(kitItemName)
                elif kitItemAmount == "":
                        kit.append(kitItemName)
                elif kitItemAmount.isdigit():
                        kit.append(kitItemName + " (" + str(kitItemAmount) + ")")
                if input("Would you like to add another item? (Y/n) ").lower() == "n":
                        lastItem = True
        return kit

        print("add extra item")

'''gets name of item'''
def getName(oldItem,greek):
        tempName = input("Name: [" + oldItem.name + "] ")
        if not tempName == "":
                newName = tempName
                Greek = re.findall(r"{([.\s\S]*?)}",newName)
                for i in Greek:
                        if i.istitle():
                                lookup = 'greek capital letter '
                        else:
                                lookup = 'greek small letter '
                        new = unicodedata.lookup(lookup + i)
                        old = ("{" + i + "}")
                        newName = newName.replace(old,new)
        else:
                newName = oldItem.name
        return newName

'''gets quantities of item'''
def getQuantity(oldItem,kit):
        if kit:
                kitText = "kit"
        else:
                kitText = ""
        validAmount = False
        while not validAmount:
                tempTotal = input("Total Amount: [" + oldItem.quantity["total"] + "] ")
                if not tempTotal == "":
                        newTotal = tempTotal
                else:
                        newTotal = oldItem.quantity["total"]
                        if newTotal == "":
```

```python
                    newTotal = str(0)
            tempService = input("Total in Service: [" + oldItem.quantity["service"] + "] ")
            if not tempService == "":
                    newService = tempService
            else:
                    newService = oldItem.quantity["service"]
                    if newService == "":
                            newService = str(0)
            tempRepair = input("Total under Repair: [" + oldItem.quantity["repair"] + "] ")
            if not tempRepair == "":
                    newRepair = tempRepair
            else:
                    newRepair = oldItem.quantity["repair"]
                    if newRepair == "":
                            newRepair = str(0)
            if int(newService) + int(newRepair) == int(newTotal):
                    validAmount = True
            else:
                    print("Totals do not add up. Please try again.")
            quantity = {}
            quantity["total"] = newTotal
            quantity["service"] = newService
            quantity["repair"] = newRepair
        return quantity


'''gets manufacturer of item'''
def getManufacturer(oldItem):
        tempMan = input("Manufacturer: [" + oldItem.manufacturer + "] ")
        if not tempMan == "":
                newMan = tempMan
        else:
                newMan = oldItem.manufacturer
        return newMan


'''gets model of item'''
def getModel(oldItem):
        tempModel = input("Model: [" + oldItem.model + "] ")
        if not tempModel == "":
                newModel = tempModel
        else:
                newModel = oldItem.model
        return newModel


'''gets location of item'''
def getLocations(oldItem,validRooms):
        print("To remove an existing location complete designate room as 'removed' ")
        newLocations = []
        for i in oldItem.locations:
                location = {}
                newRoom = validRoom(i["room"],validRooms)
                if newRoom:
                        newStorage = validStorage(i["storage"])
                        if not newRoom == "" and not newStorage == "":
                                location["room"] = newRoom
                                location["storage"] = newStorage
                                newLocations.append(location)
        allLocations = False
        while not allLocations:
                if input("Is there another location (y/N) ") == "y":
                        location = {}
                        location["room"] = validRoom("",validRooms)
                        location["storage"] = validStorage("")
                        if not location["room"] == "" and not location["storage"] == "":
                                newLocations.append(location)
                        else:
                                print("Location is invalid. Not adding location")
                else:
                        allLocations = True
```

```python
                return newLocations

'''validats room entry'''
def validRoom(oldRoom,validRooms):
        valid = False
        while not valid:
                tempRoom = input("Room: ["+ oldRoom +"] ")
                if tempRoom == "removed":
                        return False
                if not tempRoom == "":
                        if tempRoom in validRooms:
                                newRoom = tempRoom
                                valid = True
                        else:
                                print(tempRoom + " is not a valid room. Please try again.")
                else:
                        newRoom = oldRoom
                        valid = True
        return newRoom

'''gets storage room'''
def validStorage(oldStorage):
        tempStorage = input("Storage: [" + oldStorage + "] ")
        if not tempStorage == "" :
                newStorage = tempStorage
        else:
                newStorage = oldStorage
        return newStorage


'''Collects input from user on the new piece of equipment'''
def getEquipInfo(oldItem,validRooms,greek):
        info = {}
        infoIsKit,infoName,infoKit = getKit(oldItem)
        info["idnum"] = oldItem.id_num
        info["is_kit"] = infoIsKit
        info["name"] = infoName
        info["kit"] = infoKit
        if not infoIsKit:
                info["name"] = getName(oldItem,greek)
        infoQuantity = getQuantity(oldItem,infoIsKit)
        info["quantity"] = infoQuantity
        infoManufacturer = getManufacturer(oldItem)
        info["manufacturer"] = infoManufacturer
        infoModel = getModel(oldItem)
        info["model"] = infoModel
        infoLocations = getLocations(oldItem,validRooms)
        info["locations"] = infoLocations
        return info

'''displays collected information for comfirmations'''
def checkEquipInfo(info):
        print("")
        print("-----------------------------------------------------------")
        print("")
        print("ID Number: " + info["idnum"])
        print("Name: " + info["name"])
        print("Is Kit: " + str(info["is_kit"]))
        if info["is_kit"]:
                print("Kit Contents:")
                print(info["kit"])
        print("Total amount: " + info["quantity"]["total"])
        print("Amount in service: " + info["quantity"]["service"])
        print("Amount under repair: " + info["quantity"]["repair"])
        print("Manufacturer: " + info["manufacturer"])
        print("Model: " + info["model"])
        locations = info["locations"]
        count = 0
```

```python
        for i in locations:
                count = count + 1
                print("Location-" +str(count) + ": " + i["room"] + i["storage"])
        if input("Is the displayed information is correct. (y/N) ") == "y":
                return True
        else:
                return False



'''Modifies equimpnet object and adds them to new equipmentDB xml'''
def addEquip(item,info,eqdb):
        item.is_kit = info["is_kit"]
        if item.is_kit:
                item.kit = info["kit"]
        item.name = info["name"]
        item.quantity["total"] = info["quantity"]["total"]
        item.quantity["service"] = info["quantity"]["service"]
        item.quantity["repair"] = info["quantity"]["repair"]
        item.manufacturer = info["manufacturer"]
        item.model = info["model"]
        item.locations = info["locations"]
        eqdb.addItem(item)



#Functions for updating equipment images

def imgInfo(imageDir):
        images = os.listdir(imageDir)
        imageInfo = []
        ids = []
        for i in images:
                ids.append(i[:4])
                image = {}
                image["id"] = (i[:4])
                image["name"] = i
                imageInfo.append(image)
        print(ids)
        return imageInfo,ids

def outInfo(name, equipRoot):
        eqID = str(name)[:4]
        webPath = equipRoot + "/" + name

        print("??????????????????????????????")
        print(webPath)
        return eqID,webPath




#Functions for updating manuals

'''Generates list of ID numbers for all equipment with a manual'''
def equipWithMan(inputDir):
        listofID = []
        for root, dirs, files in os.walk(inputDir):
                for i in files:
                        equipID = str(i)[:4]
                        listofID.append(equipID)
        return list(set(listofID))

'''Gathers info on each of the manuals for indiviudal pieces of equipment'''
def manInfo(ID,inputDir,equipDir):
        manInfos = []
        for root, dirs, files in os.walk(inputDir):
                for i in files:
                        if ID in i:
                                dictName = {}
```

```python
                                docPath = "/".join([equipDir, i])
                                dictName["location"] = str(docPath)
                                ID = str(i)[:4]
                                dictName["name"] = nameGenerator(ID,manInfos)
                                manInfos.append(dictName)
        return manInfos, ID


'''Determines the number of manuals available for each item'''
def numOfManuals(ID,currentList):
        num = 1
        for i in currentList:
                if ID in i["location"]:
                        num = num + 1
        return num


'''Generates names for manuals based on number of manuals available for each item'''
def nameGenerator(ID,currentLst):
        manualNum = numOfManuals(ID,currentLst)
        if manualNum == 1:
                name = ID + "-manual.pdf"
        else:
                name = ID + "-manual" + str(manualNum) + ".pdf"
        return name



### Main Script

version = "1.3"


'''List of valid rooms and semesters '''
validRooms = ["ST009","ST017","ST020","ST025","ST029","ST030","ST032","ST034","ST036","ST037","ST038","ST039"
    ↪ ,"ST042","ST046","ST048","ST050","ST068","ES002", "Chem_Store", "SA_2nd_Floor", "unknown"]

'''dictionary of greek characters with matching unicode '''
greek = {"alpha": "\u03B1", "micro": "\u03BC"}

'''Define user options'''
parser = argparse.ArgumentParser()
parser.add_argument('-d', '--delete', help='Delete_a_piece_of_equipment_from_db.', action='store_true')
parser.add_argument('-e', '--edit', help='Edit_details_of_a_piece_of_equipment.', action='store_true')
parser.add_argument('-i', '--images', help='Add_all_images_in_~/staffresources/equipment/equipimg_to_the_equipment_
    ↪ database.', action='store_true')
parser.add_argument('-m', '--manuals', help='Add_all_manuals_in_~/staffresources/equipment/equipman_to_the_
    ↪ equipment_database.', action='store_true')
parser.add_argument('-n', '--new', help='Add_new_piece_of_equipment.".', action='store_true')
parser.add_argument('-t', '--test', help='Debug_mode.', action='store_true')
parser.add_argument('-x', '--validate', help='Disable_validation_for_xml.', action='store_true')
parser.add_argument('-v', '--version', help='Print_current_verion_of_script.', action='store_true')
args = parser.parse_args()
testMode = args.test
validate = args.validate


'''Paths for files'''
root = "/usr/local/master/pjl-web"
eqdbDev = root + "/dev/equipmentDB.xml"
labdbDev = root + "/dev/labDB.xml"
eqdbData = root + "/data/equipmentDB.xml"
labdbData = root + "/data/labDB.xml"
imageLocal = "/staffresources/equipment/equipimg"
imageDir = root + imageLocal
manualLocal = "/staffresources/equipment/equipman"
manualDir = root + manualLocal


'''Changes the output to a temporary file if script is run in test mode'''
if testMode:
        destXML = root + "/dev/test_equipmentDB.xml"
```

```python
        print("−−−−−−−−−−Running in test mode.−−−−−−−−−−")
else:
        destXML= eqdbDev


'''validation disabled warning'''
if validate:
        print("validation of output file has been disabled. Be Very Careful!")


'''name of host machine this scipt was written for'''
devHost=["slug","fry"]


'''Confirm that this script won't accidently run on the wrong machine'''
testHost(devHost)


'''Create pjlDB object of each of the relevent xml files'''
eqdb = pjlDB.EquipDB("/usr/local/master/pjl−web/dev/equipmentDB.xml")
labdb = pjlDB.LabDB("/usr/local/master/pjl−web/dev/labDB.xml")


'''prints version'''
if args.version:
        print("Version " + version)
        exit()


'''Checks that the development version of both key DBs are new or as new as the live versions.'''
if not checkTimeStamp(eqdbDev,eqdbData) or not checkTimeStamp(labdbDev,labdbData):
        if not checkTimeStamp(eqdbDev,eqdbData):
                print("Equipment development database is out of synce with the live version. Please update the
                    ↪ development version before continuing.")
        if not checkTimeStamp(labdbDev,labdbData):
                print("Repository development database is out of synce with the live version. Please update the
                    ↪ development version before continuing.")
        print("Exiting...")
        exit()


#
    ↪ ###############################################################################
    ↪
```

(blank area)

```python
'''calls functions for deleting equipment item'''
if args.delete:
        deleteEquipItem(eqdb,labdb)


'''calls functions for editing an existing equipment entry'''
if args.edit or args.new:
        if args.new:
                equipItem = eqdb.newItem(eqdb.new_id)
```

```python
        if args.edit:
                equipItem = getItemToEdit(eqdb)
        confirmed = False
        while not confirmed:
                equipInfo = getEquipInfo(equipItem,validRooms,greek)
                if checkEquipInfo(equipInfo):
                        confirmed = True
                        addEquip(equipItem,equipInfo,eqdb)


'''calls functions for updateing images for equipment'''
if args.images:
        images,ids = imgInfo(imageDir)
        tree = eqdb.tree
        eqTreeRoot = tree.getroot()
        for item in eqTreeRoot:
                itemID = item.attrib["id"]
                equip = eqdb.getItem(idnum=itemID)
                if itemID in ids:
                        for i in images:
                                if i["id"] == itemID:
                                        name = i["name"]
                        #equip.thumbnail = "/" + imageLocal + "/" + name
                        equip.thumbnail = imageLocal + "/" + name
                        print(equip.thumbnail)
                else:
                        equip.thumbnail = "/img/img-placeholder.png"
                        print(equip.thumbnail)
                eqdb.addItem(equip)

'''calls functions for updateing manuals for equipment'''
if args.manuals:
        print("adding_manuals")
        listOfIDs = equipWithMan(manualDir)
        print(listOfIDs)
        for i in listOfIDs:
                equipManuals,eqID = manInfo(i,manualDir,manualLocal)
                #equipManuals,eqID = manInfo(i,manualDir,"/" + manualLocal)
                print(equipManuals,eqID)
                equip = eqdb.getItem(idnum=eqID)
                equip.documents = equipManuals
                print(equip.documents)
                eqdb.addItem(equip)
# db.save("../updatedequipmentDB.xml", ignore_validation=False, error_log=True)

'''saves new xml file'''
if args.test:
        print("writing_to_" + destXML)
        print(type(destXML))
        eqdb.save(destXML, ignore_validation=validate, error_log=True)
else:
        print("saving_to_" + destXML)
        eqdb.save(destXML, ignore_validation=validate, error_log=True)

'''confirms that the script has ended properly'''
print("...and_then_there_will_be_cake")
```

6.1.4 Convert Images for Inventory Database - convertImg.py

```python
#!/usr/bin/python3

#import packages
import os

rootDir="/usr/local/master/labs/rawphotos"
output=rootDir + "/output"
```

```
imgageList = []
def listOfImg(rootDir):
        imageList = []
        print(rootDir)
        for root, dirs, files in os.walk(rootDir):
                for i in files:
                        if "img.jpg" in i:
                                imageList.append(i)

        return imageList
rotate = input("Angle_to_rotate_photos_by_[0]:_")
imageList = listOfImg(rootDir)
for i in imageList:
        inputPath = rootDir + "/" + i
        if rotate == "":
                os.system("convert_" + inputPath + "_−resize_200x200^_−crop_200x200+0+0_" + output + "/"+ i)
        else:
                os.system("convert_" + inputPath + "_−resize_200x200^_−crop_200x200+0+0_−rotate_" + rotate + "_" +
                        ↪ output + "/"+ i)
```

6.1.5   Rotate Repository Database Version - repWheel.py

```
#!/usr/bin/python3

#import packages
import os as os #docs: https://docs.python.org/3.4/library/os.html
import argparse #docs: https://docs.python.org/3.4/library/argparse.html

#parameters

root = "/usr/local/master/pjl−web/"
db_storage = root + "data/"
new_xml = root+ "dev/labDB.xml"


def get_db_files():
    all_files = os.listdir(db_storage)
    db_files = []
    for f in all_files:
        if f.startswith("labDB") and f.split(".")[0][−1] in ['0','1','2','3','4','5','6','7']:
            db_files.append(f)
    return sorted(db_files)

def increment_files(files):
    for i,f in enumerate(files):
        name = f.split(".")[0]
        index = int(name[−1])
        index += 1
        f = name[:−1] + str(index) + ".xml"
        os.rename(db_storage + files[i], db_storage + f)
    os.rename(db_storage + "labDB.xml", db_storage + "labDB−0.xml")
    os.system("cp_" + new_xml + "_" + db_storage + "labDB.xml")
    os.system("chmod_644_" + db_storage + "labDB.xml")




#MAIN

if __name__ == "__main__":
    files = get_db_files()
    increment_files(list(reversed(files)))
```

6.1.6   Rotate Inventory Database Version - eqWheel.py

```python
#!/usr/bin/python3

#import packages
import os as os #docs: https://docs.python.org/3.4/library/os.html
import argparse #docs: https://docs.python.org/3.4/library/argparse.html

#parameters

root = "/usr/local/master/pjl-web/"

db_storage = root + "data/"
new_xml = root + "dev/equipmentDB.xml"

def get_db_files():
    all_files = os.listdir(db_storage)
    db_files = []
    for f in all_files:
        if f.startswith("equipmentDB") and f.split(".")[0][-1] in ['0','1','2','3','4','5','6','7']:
            db_files.append(f)
    return sorted(db_files)

def increment_files(files):
    for i,f in enumerate(files):
        name = f.split(".")[0]
        index = int(name[-1])
        index += 1
        f = name[:-1] + str(index) + ".xml"
        os.rename(db_storage + files[i], db_storage + f)
    os.rename(db_storage + "equipmentDB.xml", db_storage + "equipmentDB-0.xml")
    os.system("cp " + new_xml + " " + db_storage + "equipmentDB.xml")
    os.system("chmod 644 " + db_storage + "equipmentDB.xml")




#MAIN

if __name__ == "__main__":
    files = get_db_files()
    increment_files(list(reversed(files)))
```

6.1.7   Check Website Links - linkCheck.py

```python
#!/usr/bin/python3

#import packages
import os
import re
import urllib.request as rq
import xml.etree.ElementTree as ET




r = re.compile(r'href="([^"]*)"')
links = []
pjlroot = "http://www.pjl.ucalgary.ca"
count404 = 0


def get_status_code(url):
    """ This function retreives the status code of a website by requesting
        HEAD data from the host. This means that it only requests the headers.
        If the host cannot be reached or something else goes wrong, it returns
```

```
            None instead.
        """
        try:
            req = rq.Request(url, headers={"User−Agent": "Mozilla/5.0_(X11;_U;_Linux_i686)_Gecko/20071127_Firefox/2.0.0.11
                ↪ "}, method="HEAD")
            conn = rq.urlopen(req)
            return str(conn.getcode())
        except Exception as e:
            return str(e.code)


def getPathsFromXML(xml_path):
    tree = ET.parse(xml_path)
    root = tree.getroot()
    paths = root.findall(".//Path")
    paths[:] = [pjlroot + i.text for i in paths]
    return paths



def checkLink(url, filename=None):
    global count404
    status = get_status_code(url)
    if status != "401":
        if filename:
            print("STATUS:_" + status + "_____for_link_" + url + "_in_file_" + dirpath + "/" + ff)
        else:
            print("STATUS:_" + status + "_____for_link_" + url)
    else:
        count404 += 1
        return


for dirpath, dirnames, files in os.walk("../../"):
    for ff in files:
        if (ff.endswith(".html") or ff.endswith(".txt")):
            with open(dirpath + "/" + ff, "r") as f:
                try:
                    s = f.read()
                    for m in re.findall(r, s):
                        if m.startswith("http"):
                            checkLink(m)
                except:
                    print("Failed_to_read_file:_", dirpath + "/" + ff)



[checkLink(i) for i in getPathsFromXML("../../data/labDB.xml")]
print(str(count404), "forbidden_requests")
```