

Educational Laboratory Website Manual

October 24, 2018

Contents

0.1	Introduction	3
0.2	Making Changes to Website	4
0.2.1	Sync Live Version and Development Space	4
0.3	Repository	4
0.3.1	Directory structure	4
0.3.2	Adding a new version of an existing lab to the repository	5
0.3.3	Adding a new lab to the repository	5
0.3.4	Repository xml template	6
0.4	Inventory	7
0.4.1	Inventory structure	7
0.4.2	Adding New Equipment	7
0.4.3	Adding photos of equipment	8
0.4.4	Deleting Old Equipment	9
0.5	Schedules	9
0.6	Scripts	9
0.6.1	Add New Lab - addNewLab.py	9

0.1 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.

0.2 Making Changes to 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. Because of this it is important to sync the development and live version of the website before making changes.

0.2.1 Sync Live Version and Development Space

The script “liveUpdate.py” (Listing 0.6.1) has been designed to sync the live version of the website with the development space for the website. It is important to run this script before and after making changes to the development space. It is run the first time to make sure that the equipment xml file in development has been updated with the changes that made directly on the pjl website. Once the changes to the development space have been made and tested the changes can be pushed to the web-server by running the same command again.

To sync run...

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

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

```
sudo ./liveUpdate.py -t
```

0.3 Repository

0.3.1 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

```

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.

0.3.2 Adding a new version of an existing lab to the repository

1. Compile a folder that contains all file needed to generate the lab experiment pdf, as well as any files need for the lab to run (ie, excel templates, vernier cmbl file, or simulations)

0.3.3 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/validDisciplines.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*

0.3.4 Repository xml template

```
<Labs>
  <Lab labId=" 0001">
    <Name />
    <Disciplines>
      <Discipline />
      ...
    </Disciplines>
    <Topics>
      <Topic />
      ...
    </Topics>
    <Versions>
      <Version>
        <Path />
        <Semester />
        <Year />
      </Version>
    </Versions>
  </Lab>
</Labs>
```

```

        <Course />
        <Directory />
    </Version>
    ...
</Versions>
<Equipment>
    <Item id="0001">
        <Name />
        <Amount />
    </Item>
    ...
<Equipment />
<Type />
<SupportDocs>
    <Doc>
        <Name />
        <Path />
    </Doc>
    ...
</SupportDocs>
<Software>
    <Name />
    ...
</Software>
</Lab>
...
</Labs>

```

0.4 Inventory

0.4.1 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.

0.4.2 Adding New Equipment

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

```
./equipmentEdit.py -n
```

A prompt will appear that will as 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 a summary will be displayed as well as a request for confirmation. If the user confirms that the information it will go through a validation process to ensure that the name is unique. If everything check out it will be added to the /dev version of the equipmentDB.xml file.

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

0.4.3 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 format.

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.

0.4.4 Deleting Old 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.

0.5 Schedules

The name of the current schedules should be

- schedule-current.pdf
- rooms-current.pdf

0.6 Scripts

0.6.1 Add New Lab - addNewLab.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
'''

import pjlDB
import os, argparse, re

#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_" + host + "_only_Exiting...")
        exit()

def checkTimeStamp(dev,data):
    '''
        Checks that the source files for the databases referenced are the latest. This protects against o

    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

'''creates 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("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("ID_format_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
    '''

```

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 abbreviations
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
,,,

```
print("Adding_version_to_" + originalItem.name + "\".")
new_version = {}
#new_version = {'originalDir': '/home/pgimby/labs/under-construction/211SP2018/lab2/', 'pdf': 'Phys 211-221 -
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("")
new_version["year"] = validYear()
new_version["directory"], new_version["labFolder"] = validExistingDirectory(new_version, originalItem, semester)
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_lab_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:
        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 abbreviations for semesters with full name

    Return:
        directory (str) full path name on new version directory
    '''
    samplePath = lab.versions[0]["directory"]
    labFolder = "/" + 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'] + "]_")
        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-n
            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_thi
                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. ")
    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)
            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 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 y/n\n"):
            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\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
        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)
        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):
    """
    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 databas

    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:
    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):
        print("E1")
        os.system("mkdir_" + versionDir)
        print("E2")
        os.system("echo_rsync_avz_—exclude_Support_Docs_" + info["originalDir"] + "_" + versionDir)
        os.system("sudo_rsync_avz_—exclude_Support_Docs_" + info["originalDir"] + "_" + versionDir)
        print("E3")
    else:
        print("E4")

```

```

        print("Lab_folder_" + versionDir + "_Already_Exists.")
        print("E5")
        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"
    print("F1")
    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)
            print("A1")
        if os.path.isdir(destinationDir):
            os.system("rsync_-avz_" + originDir + "/" + destinationDir)
            print("A2")
        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 abbreviations
        eqdb (pjlDB.EquipDB) entire equipment inventory database object generated by pjlDB
        disciplineSource (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
        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)
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.
        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
    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 abbreviation

    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

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
        disciplineSource (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
    """
    print("Editing info for \" " + originalItem.name + "\".")
    print("")
    originalItem.equipment = getEquipList(eqdb, originalItem)
    print("")
    originalItem.software = getSoftwareList(originalItem, softwareSource)
    print("")
    originalItem.disciplines = getDisciplineList(originalItem, disciplineSource)
    print("")
    originalItem.topics = getTopicList(originalItem, topicSource)
    return originalItem

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-id"])

#Main Script

version = "1.1"

'''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()
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 version of script.', action='store_true')
args = parser.parse_args()
testMode = args.test
validate = args.validate
print(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 script was written for'''
#devhost = "slug"
devhost = ["slug","fry"]

'''Confirm that this script won't accidentally 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_sync_with_the_live_version._Please_update_the_develo")
    if not checkTimeStamp(labdbDev,labdbData):
        print("Repository_development_database_is_out_of_sync_with_the_live_version._Please_update_the_deve")
    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,
    print("A")
    confirmEntry(versionInfo)
    print("B")
    if validDB(versionInfo, lab, labdb) and validDir(versionInfo, root):
        print("C")
        labdb.save(destXML, ignore_validation=validate, error_log=True)
        print("D")
        if not testMode:
            print("E")

```

```

        moveVersionDir(versionInfo, root)
        print("F")
        addSupportFolder(versionInfo, root)

    else:
        print("something_went_wrong")
        exit()

'''add 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)
            print(validate)
            #labdb.save(destXML, ignore_validation=False, error_log=True)
            labdb.save(destXML, ignore_validation=validate, error_log=True)
        else:
            os.system("echo_mkdir_" + newLabInfo["labFolder"])
            labdb.addLab(lab)
            print(validate)
            labdb.save(destXML, ignore_validation=validate, error_log=True)

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

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

#!/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

'''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"
#webserver="136.159.54.155"

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

def mountFolder(source, mountPoint, remote, option):
    fullSource = remote + ":" + source
    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...")
        gracefulExit(mountInfo)

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

def syncFolder(testMode, source, dest):
    print("syncing " + 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")
    #os.system(osTest + "rm " + dest + "/" + key + "-8.xml")

```

```

def wheel(dest, key, source, osTest):
    print("updating equipmentDB.xml")
    dbFiles = getDbFiles(dest, key)
    #print(dbFiles)
    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):
    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 + " {} \;")

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

'''checks that file a is newer than 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
    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:
    rsyncOption = "-avnz--no-l"
    osTest = "echo_"
else:
    rsyncOption = "-az--no-l"
    osTest = ""

'''Confirm that this script won't accidentally 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...")
        gracefulExit(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(rsyncOption, source, dest)

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

'''rsync webpage files'''

```

```
for i in webFiles:
    source = webSource + "/" + i
    dest = webMount + "/"
    syncFolder(rsyncOption, 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)

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

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