```
#!/usr/bin/env python

import os, sys, urllib, shutil, zipfile

# notice that this shit requires a connection to the web in order to properly work
# MAJOR TODO: remove swearings from remarks

##=====COMMANDS FUNCTIONS=====##


##=====ListRepo=====##
#This function access the content of the file
#"content.acpm" in the repo and print the structure
#of the componet that there are in the repo.
# FIXME - NEED TRY/EXCEPT
# FIXED - added try/except block
def ListRepo(): # online shit / I guess I undestood this shit
    file_url = repo_url + "content.acpm" # fucking repo_url is hardcoded, but is little what we can do
in this situation
    # so, content.acpm is an YAML
    print "Trying to access \'" + file_url + "\'..."
    try:
        # the contents of the repo are listed in the YAML
        # so, the repo contents are kinda displayed offline
        f = urllib.urlopen(file_url)
        content = f.read()
        f.close()

        yaml_obj = {} # dictionary
        # now we are going to replace the YAML library usage with our own parser
        yaml_obj = loadYAML(content)
        # as structure being hardcoded, there is a tight relation between the local stuff and the
repo, they use the same file structure
        # this shit works kinda like list(), but reading the repo YAML
        for x in structure:
            print x.upper() + ":"
            if yaml_obj[x] != None:
                for y in sorted(yaml_obj[x].iterkeys()): # why this shit must be sorted? it is sorted
for an homogeneous key display
                    print "   " + y + ": " + yaml_obj[x][y]
                print ""
            else:
                print ""
    except:
        print "!!! Error: Could not access the repository."
        sys.exit(1)

##=====List=====##
#This function list the structure of local components
def    List(): # I guess I understood this shit
    localpath = os.getcwd()
    for x in structure: # structure is hardcoded, so it will only look the appropriate folders, but I
can't see if there is an alternative to this crappy list
        mypath = localpath + "/" + x
        print x.upper() + ":"
        if os.path.isdir(mypath):
            tmpdirlist = os.listdir(mypath)
            for y in tmpdirlist: # here is were the magic is done
                if os.path.isdir(mypath + "/" + y):
                    print " ", y, "-", getDescriptionText(mypath + "/" + y)
        print ""

##=====Get=====##
#This function try to get the component from the repo
#if t find, the function dowanload the component package
#and unpack it.
#if not, returns error.
# FIXME
def Get(type, name): # online shit / I guess I understood this shit
    # smart way to do the stuff
    if type != None:
        lst = []
```

```
        lst.append(type)
    else:
        # copies the structure itself, not the reference to it (safer)
        # there's a other way to do this: list = list(structure), but unused becuase the list object
overlaps the builtin function list
        lst = list(structure)
    # bulletproof
    try:
        component = findRepoComponent(name, lst)
    # FIXME: check this shit for other errors that aren't a bad component type or web issues
    # bad component type
    except KeyError:
        print "!!! Error: invalid component type."
        sys.exit(1)
    # if not connected to the web
    except:
        print "!!! Error: could not access the repository."
        sys.exit(1)
    # "component" is a string
    if component != None:
        # my guess is this shit - "target" - is the fucking component .acpmpack URL
        target = os.getcwd() + "/" + component.split("/")[-2] + "/"
        # my guess is this shit is the fucking file name that will be on the local file system,
the .acpmpack
        fileName = component.split("/")[-1]
        try:
            download(component, target, fileName)
        except:
            print "!!! Error: Component " + name + " not found"
            sys.exit(1)
    else:
        print "!!! Error: Component " + name + " not found"
        sys.exit(1)
    print "Component " + name + " was downloaded successfully."
    # after downloading, extracts the stuff; target + fileName is the fucking path of the
downloaded .acpmpack
    Unpack(target + fileName)

##=====Put=====##
#This function was not implemented.
def Put(type, name): # online shit / trivial shit
    print "Sorry!! Function not yet implemented."
    sys.exit(0)

##=====Pack=====##
#This function pack the component or platfomrs.
#It compress all files in a acpmpack file.
# FIXME - NEED TRY/EXCEPT
# FIXED - try/except block added
def Pack(type, name): # I guess I understood this shit
    # my initial hunch is that this function gets all the components you have on your acpm local
package
    source_files = ""
    component = ""
    if type == None and name == None:
        # this if is to get ALL the stuff
        localpath = os.getcwd()
        # walks in the file system getting each source file
        for x in structure:
            # note that localpath is kinda the same, but x alters itself on each iteration, being each
one of the contents of the list structure
            # I noticed that the list structure is kinda the base of all this shit
            mypath = localpath + "/" + x + "/"
            if os.path.exists(mypath):
                source_files = source_files + x + "/ "
        # my hunch is that, in the end, source_files = "platforms/ processors/ ip/ is/ sw/ wrappers/
Makefile"
        source_files = source_files.split()
        source_files.append("Makefile")
        package_file = "all.acpmpack"

    elif type == None and name != None: # None must be the NULL of python
```

```
        # gets just one component in the "acpm package", like the mips1, or quad_mips
        component = findLocalComponent(name, structure)
        if component != None:
            source_files = component.split("/")[-3] + "/" + component.split("/")[-2] + "/"
            source_files = [source_files]
            package_file = component.split("/")[-3] + "/" + name + ".acpmpack"
        else:
            print "!!! Error: Component " + name + " not found"
            sys.exit(1) # interesting, might be as same as raise SystemExit --> nope, SystemExit ==
sys.exit(0)
    else:
        # this works just like the elif above
        lst = []
        lst.append(type)
        # bulletproof
        try:
            component = findLocalComponent(name, lst)
        except:
            print "!!! Error: invalid component type."
            sys.exit(1)

        if component != None:
            source_files = component.split("/")[-3] + "/" + component.split("/")[-2] + "/"
            source_files = [source_files]
            package_file = component.split("/")[-3] + "/" + name + ".acpmpack"
        else:
            print "!!! Error: Component " + name + " not found"
            sys.exit(1)
    #print component
    #print source_files
    #print package_file
    #sys.exit(1)
    print "Packing " + package_file + "..."
    #cmd = "tar -czf " + package_file + " " + source_files # creates .acpmpack file package_file from
source_files using zip compression
    # bulletproof
    try:
        utilities = ZipUtilities()
        utilities.toZip(source_files,package_file)
        #p = subprocess.Popen(cmd, shell=True) # interesting... guess this might be a fork
        print "Done."
    except:
        print "!!! Error: unable to pack, please try again."
        sys.exit(1)



##=====Unpack=====##
#This function unpack the component package
# FIXME
# try/except block added, and using zipfile to extract the file
def Unpack(package): # I guess I understood this shit
    if not os.path.exists(package):
        print "!!! Error: Package " + package + " doesn't exists."
        sys.exit(1)
    else:
        # this shit is the anti-pack
        print "Unpacking " + package + " ..."
        #cmd = "tar -xzmf " + package # extracts .acpmpack file package using zip compression and
keeps the ultimate updated file (-m)
        # bulletproof
        try:
            z = zipfile.ZipFile(package,"a")
            z.extractall()
            #p = subprocess.Popen(cmd, shell=True) # fork... but I still don't understand why it forks
the tar operation
            z.close()
            print "Done."
        except:
            print "!!! Error: unable to unpack, please try again."
            sys.exit(1)
```

```
##=====Create=====##
#This function create the component specified
#if exist a local copy of the template component
#the functions copy it with the new name,
#if not, the function try to get in the repo.
# FIXME - NEED TRY/EXCEPT
# FIXED - try/except block added
def Create(type, name): # calls get, so this shit is online / I guess I undertood how this works - at
least from the inside - but the result isn't obvious to me
    # might need online testing
    # there is a need to already be a fucking template in the local repo, otherwise, it will search it
online
    # so, the template is a fucking dir tree?
    list = []
    list.append(type) # there is a REAL need here for the type
    # bulletproof
    try:
        component = findLocalComponent("template", list)
        print component
    except:
        print "!!! Error: invalid component type."
        sys.exit(1)
    if component != None:
        target = os.getcwd() + "/" + type + "/"
        try:
            # my guess is this shit copies the whole filesystem tree under component to target + name
(sorry for the kinda obvious stuff)
            shutil.copytree(component, target + name)
        except:
            print "!!! Error: Component template of " + type + " not found"
            sys.exit(1)
    else:
        # here is where it will look for the template so it can be downloaded and unpacked in the
local repo
        Get(type, "template")


##=====Start=====##
#This function download the start.package
#and unpack it. This File has the structure of dir
#and the main Makefile.
def Start(): # online shit / I guess I understood this shit
    target = os.getcwd() + "/"
    fileName = "start.acpmpack"
    try:
        # gets the base acpmpack...
        download(repo_url + fileName, target, fileName)
    except:
        print "!!! Error: File start.acpmpack not found."
        sys.exit(1)
    # ...and unpacks it
    Unpack(target + fileName)
    for folder in structure:
        os.mkdir(folder)
    print "The main structure was created successfully."

##=====Repo=====##
#This function creates the "content.acpm" file.
#The function gets all components in the local structure
#and put it in a YAML file.
def Repo(): # I guess I understood this shit
    # this shit works just like list(), but put the contents in a fucking file
    localpath = os.getcwd()
    f = open(localpath + "/content.acpm", "w")
    for x in structure:
        mypath = localpath + "/" + x
        f.write(x + ":\n")
        tmpdirlist = os.listdir(mypath)
        for y in tmpdirlist:
            if os.path.isdir(mypath + "/" + y):
                f.write("   " + y + ": " + getDescriptionText(mypath + "/" + y) + "\n")
```

```
        f.write("\n")
    f.close()
    print "The file 'content.acpm' was created successfully."

##=====Help=====##
#This function shows the help contents.
def Help(): # trivial shit
    print """
  -c,  --create              create
  -g,  --get                 get
  -h,  --help                help
  -l,  --list                list
  -lr, --listrepo            listrepo
  -pa, --pack                pack
  -up, --put                 put
  -r,  --repo                repo
  -s,  --start               start
  -u,  --unpack              unpack
    """

##=====AUX FUNCTIONS=====##

##=====getDescripitionText=====##
#This function gets the content of the file
#"desc.txt" and return it
#if the file doesn't exist, it returns none
def getDescriptionText(path): # I guess I undertood pretty well how this shit works
    path = path + "/desc.txt"
    try:
        f = open(path, "r")
        desc = f.read()
        f.close()
        return desc
    except:
        return "No description provided."

##=====download=====##
#This function gets a file in the repo and copy it
#to the user local path
def download(source, target, name): # online shit / I guess I understood this shit
    # all this shit can generate exceptions, as being file operations
    webFile = urllib.urlopen(source) # opens the webfile in the local memory (kinda obvious)
    localFile = open(target + name, 'w') # creates a local file (not *that* obvious, but still obvious)
    localFile.write(webFile.read()) # writes into it (obvious)
    webFile.close()    # closes webfile (obvious)
    localFile.close() # closes local file (obvious)

##=====findLocalComponent=====##
#This function search for the component in the users local path
#if it founds, returns the path to the component
#if not, return none
def findLocalComponent(name, lst): # I guess I understood this shit
    path = os.getcwd()
    # gets the full file system tree below path, both dirs and files
    tmpdirlist = os.listdir(path)
    for x in lst: # list can be structure or a subset of structure, AFAIK
        tmpdirlist = os.listdir(path + "/" + x)
        # if there is what it is searching for in the tmplist, it tests if it is a dir and returns it
        if name in tmpdirlist:
            if os.path.isdir(path + "/" + x + "/" + name):
                return path + "/" + x + "/" + name + "/"
    return None

##=====findRepoComponent=====##
#This function search for the component in the repo
#if it founds, returns the path to the component
#if not, return none
def findRepoComponent(name, lst): # online shit / I guess I understood this shit
    # this shit works just like findLocalComponent when seen from the outside, but reads the fucking
repo yaml file and searches it
    file_url = repo_url + "content.acpm"
    f = urllib.urlopen(file_url)
```

```
    content = f.read()
    f.close()
    # here "content" is the fucking yaml
    yaml_obj = {}
    yaml_obj = loadYAML(content)

    compName = name.split(".")[0]

    for x in lst:
        if yaml_obj[x] != None:
            if compName in yaml_obj[x].iterkeys():
                # if found the fucking shit, returns its path
                return repo_url + x + "/" + name + ".acpmpack"
    return None


##=====smartGetRepo=====##
#This function gets the appropriate repo URL
#and generates the repo.cfg file, which can
#be edited to add alternative repos
# TODO: test this shit
# tested offline and with crippled internet access (a.k.a. "Unicamp-Visitante")
def smartGetRepo():
    hardcoded_repo_url = "http://www.students.ic.unicamp.br/~ra103501/acpm_repo/" # Matheus Boy
    # hardcoded_repo_url = "http://www.students.ic.unicamp.br/~ra063091/repo/" - Matheus Nagliati
    # not satisfied with this "test"
    # something like file.exists() would be much more elegant than this try/except
    try:
        f = open("repo.cfg","r")
        repo_str = f.read()
        f.close()
        repo_list = repo_str.split("\n")
        repo_list.remove("")
        try:
            for repo in repo_list:
                a = urllib.urlopen(repo + "content.acpm")
                b = a.getcode()
                a.close()
                if b == 200:
                    return repo
        except:
            # if there isn't a available repo in the repo.cfg, returns the default
            return hardcoded_repo_url
    # if there is no config file, creates it
    except:
        f = open("repo.cfg","w")
        f.write(hardcoded_repo_url+"\n")
        f.close()
        print "Repo config file created"
        return hardcoded_repo_url


##=====loadYAML=====##
# TODO: properly document this shit
def loadYAML(content):
    dic = {}
    superiorLevel = ''
    content = content.split("\n")
    # builds the dictionary
    for line in content:
        if line != "":
            # heavy wizardry \/
            [key, value] = line.split(':')
            if (key[0] == ' '):
                # removes the whitespaces before the key name
                key = key.lstrip()
                # subtree
                dic[superiorLevel][key] = value
            else:
                # tree
                dic[key] = {}
                superiorLevel = key
```

```
    # cleans the dictionary
    for l in structure:
        if dic[l] == {}:
            dic[l] = None
    return dic


##=====AUX CLASS=====##


##=====ZipUtilities=====##
# TODO: properly document this shit
class ZipUtilities:
    # this method now takes a list as arg
    def toZip(self, files_to_zip, filename):
        zip_file = zipfile.ZipFile(filename, "w")
        for file_to_zip in files_to_zip:
            if os.path.isfile(file_to_zip):
                zip_file.write(file_to_zip)
            else:
                self.addFolderToZip(zip_file, file_to_zip)
        zip_file.close()

    # aux method
    def addFolderToZip(self, zip_file, folder):
        for file_ in os.listdir(folder):
            full_path = os.path.join(folder, file_)
            if os.path.isfile(full_path):
                zip_file.write(full_path)
            elif os.path.isdir(full_path):
                self.addFolderToZip(zip_file, full_path)


##=====DEFINES=====##
# a.k.a. FUCKING HARDCODED SHIT

##=====repo location=====#
repo_url = smartGetRepo()


##=====list of supported commands=====#
commands={ # fucking dictionary -- the key is the short action and the value is the action itself
"lr":"listrepo", "l":"list", "g":"get", "pu":"put",
"pa":"pack", "u":"unpack", "c":"create",
"s":"start", "r":"repo", "h":"help"
}

##=====structure of plataform=====#
structure = ["platforms", "processors", "ip", "is", "sw", "wrappers"]


# here begins the magic... and I guess I undertood it

execline = [] # this shit is an "shortcut" to the fucking action itself -- line... would it be a queue
(FIFO)?
#get cmdline args
cmdline=sys.argv[1:] # my guess is here where the arguments are caught (this remark is redundant due
to the original developer remark above)

if cmdline==[]:
    print "!!! Error: Invalid action."
    sys.exit(1)
elif len(cmdline) > 3: # this is kinda a magic number...
    print "!!! Error: Too many arguments"
    sys.exit(1)


cmd = cmdline[0]
#identify the command and add it to an execution line
if cmd[0:1]=="-" and cmd[1:2]=="-": # test for the --command
    values = commands.values()
    if cmd[2:] in values:
```

```
        execline.append(cmd[2:])
    else:
        print "!!! Error: " + cmd + " is an invalid action."
        sys.exit(1)

elif cmd[0:1]=="-" and cmd[1:2]!="-": # test for the -cm
    if commands.has_key(cmd[1:]):
        execline.append(commands[cmd[1:]])
    else:
        print "!!! Error: "+cmd+" is an invalid short action."
        sys.exit(1)
else:
    print "!!! Error: " + cmd + " is an invalid action."
    sys.exit(1)


execline.extend(cmdline[1:]);
action = execline[0]
# execline:
# [0]: action
# [1]: type
# [2]: name

if len(execline) != 1:
    if len(execline) == 2:
        type = None
        name = execline[1]
    else:
        type = execline[1]
        name = execline[2]
else:
        type = None
        name = None


#switch with the functions calls
if action == "listrepo":
    ListRepo()

elif action == "list":
    List()

elif action == "get":
    Get(type, name)

elif action == "put":
    Put(type, name)

elif action == "pack":
    Pack(type, name)

elif action == "unpack":
    Unpack(name)

elif action == "create":
    Create(type, name)

elif action == "repo":
    Repo()

elif action == "start":
    Start()

elif action == "help":
    Help()

else:
    print "!!! Error: No command found" # how the hell am I going to get this error?
    sys.exit(1) # maybe this is an equivalent to "raise SystemExit", or "raise SystemExit" is syntatic
sugar for this?
```

```
# Matheus Boy's remarks:

# The code is kinda well written and clear, but - allowing myself to be an ass - my guess is because
the developer wasn't really fluent of python
#(neither I fully am) so, he didn't use heavy wizardries; also, I was already familiar with python, so
it was easier to undertand the code.

# I've got kinda interested in some issues I found along the study of the code. The main issue is that
the user *needs* to have a connection to the web in order
# to this stuff work 100% well and trigger no errors from the python interpreter. Some stuff that is
online could be in try/except blocks so it screams
# no ugly python errors.

# I couldn't get to run the quad_sort software, because I need other "acpm tools" to do it. The
problem isn't in this code, but in the Makefile. There is
# some hardcoded stuff in Makefiles that can lead to some trouble. I don't know if it was an spec of
the project to make ALL things work, but I think it is a
# point of interest, and may need further care.
```