Automatically Archive Files on the Linux cluster at the Boston Fed

Jamie Fogel

May 21, 2014

1 The Problem

At the Boston Fed we do not have any sort of version control system. This means that once you update and save a do-file (or other type of program) the old version is lost. While the Linux cluster is constantly backed up, providing some opportunity to restore old files, this process may be slow and requires you to know exactly which files you wish to restore. An ideal system would:

- Store all of your old files with a time stamp so that you have access to and a record of previous versions
- Be stored locally so that you can access your old files yourself rather than requesting a restore through CLS
- Not take up enough file space to create a burden on the system
- Be sufficiently automated that the user does not need to make a conscious effort to use the system

I believe that I have created a system that comes close enough to accomplishing the above so as to be useful to the entire department.

2 The Python Script

My solution consists of a relatively simple python script that can be used in conjunction with **cron**, a Unix time-based job scheduler. **Cron** allows you to schedule certain unix jobs to run at specified intervals[1] The python script, **archive.py**, loops over all files within the current working directory and all of its subdirectories. It checks whether or not each file has been modified today and, if not, creates a time-stamped copy of that file within a directory called "archives" located within the parent directory (**archive.py** will create the "archives" folder when it is first run). My script only archives .do, .ado, .py, .m, .r, .R, .tex, and .sas files, however other file types can easily be added. I

[1] For more details see http://en.wikipedia.org/wiki/Cron.

include these file types because they are commonly used and relatively small. This program should only be used to archive script files, as attempting to archive data and other files may very quickly use an excessive amount of file storage.

All archived files will be named according to the following convention:

- All slashes ("/") are converted to double dashes ("–").
- A time stamp of the form "-YYYY-MM-DD" is appended to the end of the file name before the file extension.

For example, if I run archive.py from my home directory (`/home/homedir/`) a file stored as `/home/homedir/subdir/file.do` and modified on May 21, 2014 will be archived as `/home/homedir/archives/subdir--file-2014-05-21.do`.

Finally, a log is kept of each run of archive.py in a file called "archivelog.txt" that is created in your home directory. Be sure to check this file regularly to be sure that the archive process is executing properly.

3 Using cron to Run the Python Script at Specified Intervals

This part is somewhat complicated but only needs to be done once. Just follow these steps:

1. You must request permission to edit your **crontab** file from Jones.
2. Open the terminal on the cluster.
3. Type "crontab -e" in the terminal command line to open your crontab file.
4. Edit your crontab file to specify the intervals at which you want to run the archive script. You specify jobs as follows:
   min hour day_of_month month day_of_week command to execute.
   My crontab file looks like this:

   ```
   00 23 * * * cd /home/homedir/   &&      python /home/homedir/archive.py
   00 23 * * * cd /shared/bpsdata/ &&      python /home/homedir/archive.py
   00 23 * * * cd /data/CPS        &&      python /home/homedir/archive.py
   00 23 * * * cd /data/SIPP/      &&      python /home/homedir/archive.py
   ```

   This runs my script every night at 11PM. I specify that I want to run the script in four different working directories. This means that I will save archived files in four different places: /home/homedir/archives/, /shared/bpsdata/archives/, /data/CPS/archives/, and /data/SIPP/archives/.

   For some reason, you must edit your crontab file using vim commands. This is somewhat complicated so if you are unfamiliar with vim I suggest you Google it or ask Jones or someone else for help.

4 Disclaimer

This script has not been extensively tested and I am not very experienced in python or unix, so there may be errors. However, I have been using this system for five months without issue. If you encounter any problems please feel free to contact me at jamie.fogel@gmail.com.

Your crontab setup may become broken when linux cluster maintenance occurs, so be sure to check that archives are still being created after maintenance.

5 archive.py

Below is the code for archive.py:

```
"""
archive.py
Jamie Fogel, Dec 18, 2013

Adapted from code written by Vikram Jambulapati to automatically archive old do-
    files. My code gets all files
        of a specified file type within a particular file tree. This version saves
            all archives in a folder
        called 'archives' in the parent directory.
This file should be saved in your home directory and called from the parent
    directory you wish to archive.
        For example, if I want to archive /home/homedir/, then /home/homedir/ must
            be my working directory when
        archive.py is called

"""


#!/usr/bin/python
import os
import shutil
import datetime
import re

# Define base path
base_path = os.getcwd()

# Get current date (full time stamp) and convert to YMD string
current_date = datetime.datetime.now()
current_ymd = "%02d-%02d-%02d" % (current_date.year, current_date.month,
    current_date.day)

# Compile regular expressions for later use
archives = re.compile("/archives$")
do = re.compile(".do$")

# Ensure that a archives folder exists, if not create one
archive_path = os.path.join(base_path,"archives")
if not os.path.exists(archive_path):
```

```
    os.makedirs(archive_path)

# Function creates a new file name for files to be archived. Full filepath of
    original file included in
#        filename with "/" changed to "--" and timestamp appended to end of
    filename.
def new_path(old_path, base_path):
    # Get file timestamp (Unix), convert to Python time, and convert to YMD string
    file_time_unix = os.path.getmtime(old_path)
    file_time_human = datetime.datetime.fromtimestamp(int(file_time_unix))
    file_time_ymd = "%02d-%02d-%02d" % (file_time_human.year, file_time_human.
        month, file_time_human.day)

    temp_path = "%02s-%02s%02s" % (os.path.splitext(old_path)[0], file_time_ymd,
        os.path.splitext(old_path)[1])
    tempstr = base_path+"/"
    temp_path = temp_path.replace(tempstr, "")
    temp_path = temp_path.replace("/", "--")
    new_path = base_path+"/archives/"+temp_path

    return {'new_path': new_path,'file_time_ymd': file_time_ymd}

# Function checks whether or not a given file has already been archived
def already_archived(root, file, archive_folder):
    old_path = os.path.join(root, file)
    pattern = "%02s-[0-9]+-[0-9]+-[0-9]+%02s" % (os.path.splitext(old_path)[0], os
        .path.splitext(old_path)[1])
    pattern = pattern.replace("/", "--")
    pattern = pattern.replace(".do","\.do")
    base = os.getcwd()
    base = base.replace("/","--")
    base = base+"--"
    pattern = pattern.replace(base, "")

    for s in os.listdir(archive_folder):
        regex = re.compile(pattern)
        """print ""
        print s
        print pattern
        print regex.search(s)"""
        if regex.search(s)!=None:
            return True
    return False

# Function checks whether file is of type to be archived as defined by 'filetypes'
     list
def check_file_type(file, filetypes):
    for f in filetypes:
        if os.path.splitext(file)[1]==f:
            return True
    return False

# Functions checks whether file is in the .Trash folder
def check_trash(root):
    pattern = ".*/\.Trash.*"
    trash_regex = re.compile(pattern)
    if trash_regex.search(root)!=None:
        return True
    else:
```

```
        return False


# Define file types to be archived
filetypes = ['.do', '.ado', '.py', '.m', '.r', '.R','.tex', '.sas']

# Loop over all directories and subdirectories
for root, dirs, files in os.walk(base_path):
    if not archives.search(root):

        for file in files:
            # Only copy do-files
            """
            if file[0] == ".":
                continue
            elif not do.search(file):
                continue
            else:
            """
            # Only archive files of specified type
            if check_file_type(file, filetypes)==True and file[0]!="." and not
                check_trash(root) :
                old_path = os.path.join(root, file)

                # Get file timestamp (Unix), convert to Python time, and convert
                    to YMD string
                file_time_unix = os.path.getmtime(old_path)
                file_time_human = datetime.datetime.fromtimestamp(int(
                    file_time_unix))
                file_time_ymd = "%02d-%02d-%02d" % (file_time_human.year,
                    file_time_human.month, file_time_human.day)

                archive_path = base_path +"/archives"

                # If file has been modified today, create new archive
                if current_ymd == new_path(old_path, base_path)['file_time_ymd']:
                    shutil.copy2(old_path, new_path(old_path, base_path)['new_path
                        '])
                    print "Copied %s to %s" % (file, new_path(old_path, base_path)
                        ['new_path'])

                # If no version of the current file has ever been archived,
                    archive
                elif not already_archived(root, file, archive_path):
                    shutil.copy2(old_path, new_path(old_path, base_path)['new_path
                        '])
                    print "Copied %s to %s" % (file, new_path(old_path, base_path)
                        ['new_path'])

                else:
                    print "%s already archived" % (file)



current_date = datetime.datetime.now()
current_ymd  = "%02d-%02d-%02d" % (current_date.year, current_date.month,
    current_date.day)
current_time = "%02d:%02d:%02d" % (current_date.hour, current_date.minute,
    current_date.second)
```

```
print current_ymd

archivelog = os.getcwd()+'/archivelog.txt'
line = "Ran on %s at %s \n" % (current_ymd, current_time)
```