AOS 452 – Lab 9 Handout
Automated Plot Generation

INTRODUCTION
The command that is new to this lab is **crontab**. **Crontab** allows one to run scripts automatically without having to be at the computer terminal to execute them. The commands needed to run the scripts are stored in a special crontab file. The UNIX system utility *cron* reads the crontab file and runs the specified commands.

The process of automatically creating web-accessible GEMPAK plots consists of four steps:

1. Prepare the script(s) (Lab 8, with some twists)
2. Prepare the crontab file (new)
3. Load the crontab file into the *cron* utility (new)
4. Code up some HTML to point to the generated plots (Lab 7)

STEP 1: PREPARING THE SCRIPT
Before using **crontab** to run a script, you'll need to add a few extra lines of text to your script. These lines go between `#!/bin/csh -f` and the line on which you specify what GEMPAK program you wish to run first (e.g., `gdplot << EOF`).

First, you need to let the *cron* utility know the location of all the executables (programs, etc.) you intend to run in the script. To do this, enter the following as the second line in your script:

`source ~`*yourusername*/.tcshrc

Next, you need to change into the directory in which the *cron* utility should be working. This could be, for example, your home directory, or your public_html directory. For the purposes of this lab, you'll want to move into your public_html directory. Any plots created by your script will appear in this directory. (*Tip: typing pwd into a fresh terminal window will give you the appropriate path for the directory you are in.)* You can change to the appropriate location by adding a line similar to the following to your script:

cd /ef5/raid6/class/fall16/*yourusername/*public_html

Finally, remember to always include **gpend** at the end of your scripts. System crashes have occurred in the past when people have failed to put **gpend** in scripts that they have used with crontabs.

Here's an example. Let's say you wanted to produce a GIF image with a white background of 12-hour forecasted 500-mb geopotential height from the 18Z 4 October 2017 GFS model run using a crontab. To do this, you could produce a script called, say, *crontabtest.csh* that looks similar to the following:

```
#!/bin/csh –f

source ~username/.tcshrc
cd /ef5/raid6/class/fall16/username/public_html
```

#This next section specifies that the file should be created with a white background

```
gpcolor << EOF1
DEVICE   = gif|500mbhght.gif
COLORS   = 101 = 255:255:255

run

exit
EOF1

gdplot << EOF
  GDFILE     = /weather/data/gemdata/hds/17100418_avn003.gem
  GDATTIM = f12
  GLEVEL    = 500
  GVCORD   = pres
  PANEL      = 0
  SKIP          =
  SCALE       = 0
  GFUNC      = hght
  CTYPE      = c
  CONTUR   = 3/1
  CINT         = 60
  LINE         =3/1/2/1
  GVECT     =
  WIND       =
  TITLE       = 1/-2/@ heights(m) valid ~
  TEXT        = 0.8
  CLEAR     = y
  GAREA     = top--
  PROJ       = utm
  MAP         = 8/1
  DEVICE   = gif|500mbhght.gif    (same gif file specified above for gpcolor)

run

exit

EOF
gpend
```

Details that you have to get right when preparing the script

A) You must have `source` and `cd` on separate lines. The script will not run correctly (if at all) if this is not done correctly.

B) The script you plan to run in a crontab must be an **executable** file. Recall that you can make scripts executable by entering the following at a UNIX prompt:

chmod 700 crontabtest.csh

To confirm that your script is executable, type `ls -l` (for "long" listing) at the UNIX prompt. You'll see something similar to this:

-rwx------. 1 mmmadsen fall12 966 Oct 6 11:04 crontabtest.csh

The first chunk of text lists the permissions of each file in the directory. The script is executable if there is an `x` in the fourth space from the left. Otherwise, it is not executable. In this case, *crontabtest.csh* is executable. Depending on your account settings, an asterisk (*) may appear after the filename; this also indicates the file is executable.

C) Make sure the first line of the script is `#!/bin/csh -f` so that UNIX can tell the file is a C-shell script.

D) If you wish to make GIF images with your script (to post on your website), you must use `device=gif|filename.gif` as shown in the example. Using `device=gf` will not work when your script is run by the *cron* utility, although it works fine otherwise. You can view your images with the `xv` command instead of ghostview (ggv), as well.

E) Make sure your script is working properly before moving on to Step 2. Also, delete any files (e.g., GIF images) that were created during your testing so the crontab can start fresh.

STEP 2: PREPARING THE CRONTAB FILE

Now that you have the script fully prepared, you can prepare the crontab file. ***Note that the crontab file is not the same thing as your script file(s)***. The contents of the crontab file should follow this general format:

```
A B C D E /ef5/raid6/class/fall16/username/crontabtest.csh > /ef5/raid6/class/fall16/username/crontabtest.out 2>&1
```

1 **2** **3a** **3b** **3c**

So what does this mean? Follow along on the next page to see what groups 1, 2, 3a, 3b, and 3c mean.

Group:

1) Time to run script

A Minute (0–59)
B Hour (0–23)
C Day of Month (1–31)
D Month (1–12)
E Day of Week (0–6, where 0 = Sunday, 1 = Monday, etc.)

When an asterisk appears in a field instead of a number, the *cron* utility interprets that as a wild card for all possible values. You must have spaces between A and B, B and C, etc.

Examples

`0 3 16 10 2` Run at 3:00 AM on Tuesday, October 16 (This would be about once every seven years)

`0 16 16 * *`	Run at 4:00 PM on the 16th of every month, regardless of which day of the week it is
`15 2 * * *`	Run at 2:15 AM every day
`30 20 * * 3`	Run at 8:30 PM every Wednesday, regardless of the date
`30 20 * * 2,4`	Run at 8:30 PM every Tuesday and Thursday, regardless of the date

2) Which script to run --- In this position, you list the ***full*** pathname of the script you wish to have the *cron* utility run.

3) Output redirection --- To help with debugging, you'll want to produce an output file containing all program and error messages that occur while the script is running. This part of your crontab file sets up such a file for output.

a) Output redirection symbol
The > symbol tells *cron* that output should be redirected to a particular location or file.

b) Output file
The ***full*** pathname of the output file must be given to the right of the output redirection symbol. Text generated by the program(s) run in the script will be written into this file. Although you can name this file whatever you want, I suggest using the same name as your script, but with a `.out` extension, so that you can easily find it.

c) Error message redirection
The character sequence `2>&1` tells *cron* to redirect error messages to the output file specified in 3b as well.

As an example, let's say you want to run the script *crontabtest.csh* using **crontab**. You could create a crontab file named *crontabtest.cron* that looks like the following: *(Unless you have crontabtest.csh in another location)*

```
50 14 * * * /ef5/raid6/class/fall14/username/crontabtest.csh > /ef5/raid6/class/fall16/username/crontabtest.out 2>&1
```

50 14 * * *	Run at 2:50 PM every day
/ef5/raid6/class/fall16/username/crontabtest.csh	Run the file *crontabtest.csh* located in the **/ef5/raid6/class/fall14/username** directory
>	Redirect the output
/ef5/raid6/class/fall16/username/crontabtest.out	Text output by the program(s) run by the script *crontabtest.csh* will be written into a file named *crontabtest.out* in the **/ef5/raid6/class/fall16/username** directory
2>&1	Error messages produced during the execution of the crontab will be written into the same output file as above

Details that you have to get right when preparing the crontab file

A) You must have at least one space between each field in the crontab file.

B) The text for each crontab entry must be on one line. If it is more than one line, *cron* will fail to read lines two through whatever. The word wrap feature of **gedit** may cause some of your command line to be written on a second line, so turn it off using the Preferences menu. You can also make your **gedit** window very wide.

C) You can save your crontab file with any name. However, it may be wise to use a `.cron` extension so you can easily identify the file as a crontab file when you look through your directories in the future.

STEP 3: LOADING THE CRONTAB FILE

Now that you have an executable script and a corresponding crontab file ready, you can load the crontab file into the *cron* utility.

Loading the crontab file is as simple as typing `crontab name_of_file.cron`, where *name_of_file.cron* is the name of the crontab file, not the name of your script.

To verify that your crontab file is loaded, type `crontab -l` (the letter). If you wish to unload your crontab file from the cron utility, type `crontab -r`

NOTE: Each computer in the classroom is running its own *cron* utility. Thus, if you load a crontab file while logged into luis, log out, and then log into ivan, typing `crontab -l` will not show anything. If you wanted to see your crontab or remove it, you'd have to log back into luis. So, **remember which machine you are logged into when you are loading a crontab file!**

FURTHER NOTE: Since the *cron* utility only runs in Linux, any crontab file you have loaded on the machines in 1411 will only run if that machine is running Linux at the time your job is scheduled to run. Thus, if you have a vital script that must run on schedule (e.g., something for your weather discussion), you should log into cat5 before loading your crontab file, since cat5 is never rebooted into Windows. To do this from a terminal window on a 1411 machine, type:

ssh -Y username@cat5.aos.wisc.edu

Enter your password when prompted and then follow the instructions below.

As an example, let's say you wanted to load the crontab file *crontabtest.cron* from above. The loading procedure would go as follows:

1) Type `crontab crontabtest.cron`
2) To see if *crontabtest.cron* has been properly loaded, type `crontab -l`. You will see the following on the screen:

```
50 14 * * * /ef5/raid6/class/fall16/username/crontabtest.csh > /ef5/raid6/class/fall16/username/crontabtest.out 2>&1
```

You would want to load the crontab file before 2:49 PM to be assured that the *cron* utility reads your commands in time.

3) If you wanted to unload the crontab file from the *cron* utility, you would type `crontab -r`.

Details that you have to get right when loading the crontab file

A) If the text for each crontab command is not on one line, an error message saying the commands in the crontab cannot be recognized will appear when you try loading the crontab file into the *cron* utility.

B) If you are testing a crontab file, make sure that you load it at least 60 seconds before the execution time listed in the crontab file. To see the current time, type `date` into a terminal.

TROUBLESHOOTING: AFTER THE CRONTAB IS EXECUTED

About 30–60 seconds after the scheduled time for the execution of your crontab, you should find an output file and whatever files you created from the script (PostScript, GIF, etc.) in the directory(ies) you specified. (Recall that the output file is specified in the crontab file, but the GIF or PostScript files are specified in the C-shell script.)

If the output and other files are not present:

1) Check the time in your crontab file and on the machine. You may have been too late when loading the crontab, or the time has not yet been reached when the crontab is programmed to run.
2) Your script may not be executable. Did you use the **chmod** command on your script?
3) You may be looking in the wrong directory. Type `pwd` at the prompt to see in which directory you are currently located. Check your crontab file and script to see in which directory(ies) your files should be located.

If the output file is present, but not the PostScript or GIF file(s):

1) View your output file using **gedit**. You may find one or more error messages in the window. Here are some examples:
    a. Cannot execute (This means you need to make your script executable.)
    b. Command not found [This means your script has syntax errors. Do you neglect to use proper spaces (e.g., `cd/ef5/…` instead of `cd /ef5/…`)? Perhaps you have some bad commands in your script (e.g., `gdcontour` instead of **gdcntr**).]

If all files are present in the directory, but the PostScript or GIF image looks strange:

1) Check your script for errors. It is highly recommended that you run your script yourself before running it in a crontab to assure yourself that any plotting irregularities are not due to errors in the script.
2) Sometimes images can come out strange if your crontab is run at exactly the same time as another person's crontab on the same machine.

If the crontab does not run properly, delete all files created from the erroneous run. After deleting the files, make the necessary changes to the crontab file and/or script. After the changes are completed, reload the crontab file.

RUNNING MULTIPLE CRONTAB COMMANDS

Only one crontab file (e.g., *crontabtest.cron*) can be loaded into the *cron* utility per user. However, the file can contain more than one command. Let's say you wanted to run three example files, *hght500.csh* at 6:00 AM, *sfcpres.csh* at 2:00 PM, and *absvort.csh* at 5:30 PM everyday. The crontab file would look something like this:

```
0 6 * * * /ef5/raid6/class/fall16/username/hght500.csh > /ef5/raid6/class/fall16/username/hght500.out 2>&1
0 14 * * * /ef5/raid6/class/fall16/username/sfcpres.csh > /ef5/raid6/class/fall16/username/sfcpres.out 2>&1
30 17 * * * /ef5/raid6/class/fall16/username/absvort.csh > /ef5/raid6/class/fall16/username/absvort.out 2>&1
```

These three commands will be run at the specified times when the crontab file containing these commands is loaded into the *cron* utility.

If a crontab runs that generates files with the same name as files already in existence in your directory, **the old files will be overwritten, except for GIF files**. For instance, say you run *crontabtest.csh* on three days. That script is designed to create a GIF image called *500mbhght.gif*. A listing of the files in the directory after the script runs for three days would show one output file (*hght500.out*) and three GIF files (*500mbhght.gif*, *500mbhght.gif.1*, and *500mbhght.gif.2*).

There may be situations where you want to run a script every day, but you only want the GIF image from the most recent run of the script in your directory. Such a task can be easily accomplished by adding another line to the .csh file. That line should tell the script to remove the old file before a new file with the same name is created. For example, you may want to remove the version of *500mbhght.gif* created yesterday before you run *crontabtest.csh* today. As such add the following line to crontabtest.csh:

```
#!/bin/csh –f

source ~username/.tcshrc
cd /ef5/raid6/class/fall16/username
rm 500mbhght.gif
```

This process of removing old files before creating new files will be very useful when working with images on your web pages.

STEP 4: CODING UP THE HTML

This step simply entails adding an appropriate set of `<a>` tags to your HTML. For the example we've been looking at so far, this would be something like `<a href="500mbhght.gif">500-mb heights</a>`. See Lab 7 for more details.