

Overview of Market USA FCU's Nightly Automation Process

Part 1

What we accomplished and how:

At Market USA FCU we had to solve a problem of high IT staff turnover due to the work schedule. Our daily end of day IT duties would require someone to stay late like most Ultradata credit unions. The main reason for a person to be here was to complete a backup, because although Ultradata has job scheduler for FINISH and START, they had no way to automate splitting mirrors. We were able to successfully implement a solution that automates these duties. The best benefits of this accomplishment are:

- Saved us from losing employees due to scheduling
- Alleviated the need to hire someone specifically for this job
- Eliminated user error during the end of day process
- Allowed us to run FINISH & START 7 days a week without staff here on weekends
- Provided an excellent way to automate many other jobs at the CU

The process developed consists of several scripts each completing a certain task in Market USA FCU's end of day jobs. These scripts utilize the free program *expect* designed to start unix programs and then interact with them from a script. It is a very easy scripting language to learn. The basic underlying principle of the scripts is to "expect" a certain string of characters like 'login:' and then to "send" the answer. Additionally a great utility comes with it that automates the script writing process too!

The program *expect* was written by Don Libes while working at NIST so it is completely free for anyone to use. You can get more information and download the program here <http://expect.nist.gov/>. Included in the download are several example scripts and *autoexpect*, the tool to automate script writing. Using these tools it is very easy to create your own scripts automating almost everything that needs to be done on your unix server.

We started to create our scripts using *autoexpect*. You run the program and it gives you a unix shell from which you start the process you want to automate. To simplify things all of the scripts start by initiating a telnet session from unix server to itself. This way the interaction is really like a user telneting into the server. It records all the text you see and all of your answers that you type. When you are finished, your automatically created script is located in the script.exp file. We then took the script.exp file and edited so that only the actual users prompts are what is expected. This is necessary because *autoexpect* records everything that the user sees as what should be expected. This can include many unnecessary things that are not actual user prompts. Plus it could cause the script not to run properly if for example the current date was on a screen the user saw and the next day you run the script it is expecting the previous day's date.

After we had the scripts created and cleaned up we added pager support. The *expect* command used in the scripts has a feature called timeout. After a period of time that you set, if the script does not receive the text it is expecting then it will follow the actions you specify in the

timeout. We set it up to page us with a number that represents which process has the problem. When a timeout occurs it calls a unix script which uses the free g-kernel program to dial our pagers. Information on g-kernel can be found at <http://www.columbia.edu/kernit/gkernel.html>. After the pages are sent the script will wait for an indefinite amount of time allowing for us to connect to the server and fix the problem.

Finally we created a unix script that ran each *expect* script in order and then scheduled it using cron. We scheduled it so that each weeknight FINISH is run, mirrors are split, 2 backups are created, and START runs in the morning. On the weekends we only run FINISH and START, no backups.

Part 2.

The scripts and what they do:

Script: initcheck.exp

What it does: Initiates a check printer for internet and telephone banking checks

In order to run FINISH certain things must be completed first. Most are completed after each staffer finishes their job and leaves for the day, like teller banking. Before the last IT staffer leaves they make sure FINISH can be run in ADMIN99. The only thing that can change is if members request checks through internet or telephone banking. Of course the IT staff can initiate a check printer before they leave, but this script is so that on Saturdays and Sundays no one has to be here to do this. This is scheduled in cron to run before the finish script each day.

Script: runfinish.exp

What it does: Runs FINISH

We use a script for this instead of job scheduler because it gives us more flexibility with our responses, error handling, and easy pager support. Also when we were testing backoffice job scheduler it always had errors running FINISH.

Script: cutoff.exp

What it does: Stops the processes for eft, shared branch, and hb/vr

Before you split mirrors all processes must be stopped. All of these are started and stopped by the user cute.

Script: jsoff.exp

What it does: Stops the job scheduler processes

The job scheduler processes have to be stopped also and they are started and stopped by a different user than cute, hence the two scripts.

Script: splitmirr.exp

What it does: This process stops the fsp services, stops the ud database, splits mirrors, starts the database, and starts the fsp services

This is the key step for automation. You can't use job scheduler to do this since it has to be stopped at this point.

Script: cuteon.exp

What it does: Starts the previously stopped eft, shared branch, and hb/vr services

Just starts them back up and checks to make sure they are running before finishing.

Script: json.exp

What it does: Starts the job scheduler service

The last thing to start back up. It needs to start JS because we have several things for JS to do overnight.

Script: bakup.exp

What it does: Creates 2 tape backups

This process will create 2 tape backups for us. We have two dlt tape drives to make this work. Soon we will have it create an offsite backup too.

Script: PageForHelp

What it does: Pagers support

This unix script accepts the error number as an argument and then using g-kermid dials our pagers, waits a few seconds and sends the number.

Summary

Automating these tasks has been a wonderful success for us. We have been using this process for over a year now without any trouble. This has greatly helped us keep IT staff happy and spending time supporting users not computer programs.