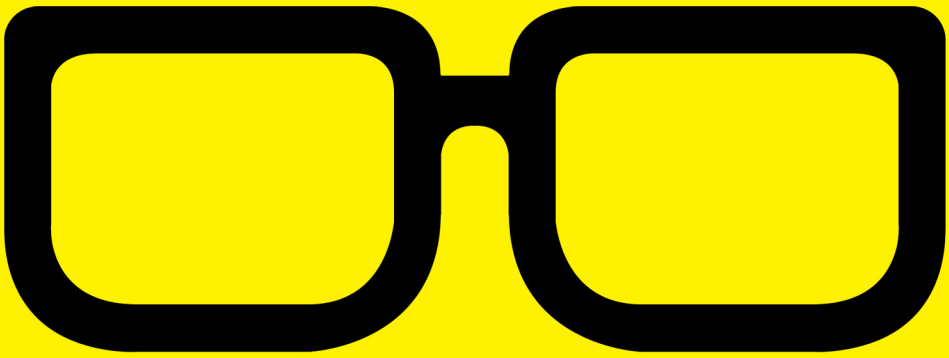
SPONSORED BY

GEEK GUIDE



Beyond Cron

How to Know When You've Outgrown Cron Scheduling—and What to Do Next

LINUX™ JOURNAL

Table of Contents

MIKE DIEHL has been using Linux since the days when Slackware came on 14 5.25" floppy disks and installed kernel version 0.83. He has built and managed several servers configured with either hardware or software RAID storage under Linux, and he has hands-on experience with both the VMware and KVM virtual machine architectures. Mike has written numerous articles for *Linux Journal* on a broad range of subjects, and he has a Bachelor's degree in Mathematics with a minor in Computer Science. He lives in Blythewood, South Carolina, with his wife and four sons.

GEEK GUIDES:
Mission-critical information for the most technical people on the planet.

About the Sponsor

Skybot, A Division of HelpSystems

HelpSystems has more than 30 years of experience in providing enterprise scheduling and automation solutions. Part of the HelpSystems family of brands, Skybot provides an affordable solution for cross-platform enterprise job scheduling, allowing businesses to integrate workflows across servers and critical business applications and monitor them from a central interface. Skybot Scheduler incorporates your disparate job schedules to help you build a unified enterprise schedule based on cross-server dependencies. For cron users, in particular, Skybot Scheduler allows you to import existing UNIX crontab data and use the cron expression to schedule new jobs using familiar cron syntax, helping to connect your UNIX cron job scheduling to enterprise operations. Skybot Scheduler also includes reporting, auditing and security capabilities to ensure that your enterprise job schedule is well documented and reliable.

For more information on Skybot Scheduler, see www.helpsystems.com/skybot.

Beyond Cron

How to Know When You've Outgrown Cron Scheduling—and What to Do Next

MIKE DIEHL

If you've spent any time around UNIX, you've no doubt learned to use and appreciate cron, the ubiquitous job scheduler that comes with almost every version of UNIX that exists. Cron is simple and easy to use, and most important, it just works. It sure beats having to remember to run your backups by hand, for example.

But cron does have its limits. Today's enterprises are larger, more interdependent, and more interconnected than ever before, and cron just hasn't kept up. These days, virtual servers can spring into existence on demand. There are accounting jobs that have to run after billing jobs have completed, but before the backups run. And, there are enterprises that connect Web servers, databases, and file servers. These enterprises may be in one server room, or they may span several data centers.

If you're like most system administrators, you've worked around these limitations by leveraging other tools like scripting languages, configuration management, SSH/scp, and so on. The results usually work, but sometimes they are hard to manage. Eventually, you have to ask yourself if you've outgrown cron. Let's take a quiz, just for fun.

Have you ever:

- Distributed a crontab file to multiple servers via `scp`, only to have to update those files later on?
- Forgotten which column you were editing and inadvertently scheduled a daily job to run every hour or vice versa?
- Had a job start before the previous instance of the same job had finished from the last run?
- Had to explain to your manager what jobs ran where and when, via cron?

- Received 15 e-mail messages in one morning telling you that everything ran properly the night before, only to miss the one message warning of an impending disk failure?

- Written a shell script or Makefile in order to manage which jobs run on which servers?

- Tried to schedule a cron job to run on the last day of the month...no matter which month?

- Allowed a junior system administrator access to your server's crontabs and hoped he or she didn't break something?

- Become proficient with cron, only to have the accounting department deploy a Windows server, with its own job scheduling mechanism?

- Put your crontabs under some kind of version control?

There's no prize for answering all of these questions correctly, but you do get points for honesty. If you found yourself answering "yes" to a number of these questions, or at least relating to the situation they describe, you may be needing to rethink your job scheduling infrastructure. Your organization may be deep enough that people of varying skill levels may be looking at, or changing, job schedules. Or, your organization may be wide enough that you have to maintain many different types of

services, such as database, Web, file, and authorization services. Your organization may be both wide and deep, in which case, you have almost certainly outgrown cron.

Today's enterprise is different from what it was even ten years ago. Let's take a look at some of the differences and discuss how an ideal enterprise job scheduling tool might deal with them.

Ease of Use

Nobody ever has said that cron was difficult to use. But let's face it, we live in a point-and-click world now. Editing a raw ASCII file with a simple editor like vi or nano is an easy way to make a simple mistake that is often difficult to spot.

It's very easy simply to press the wrong key on your keyboard. People who have worked with computers for some time and have become "touch typists" sometimes make mistakes without even knowing it. Another class of common error is simply losing track of which column in the crontab file is the day of week (DOW) and which is the hour column. Most crontab files include headers for each column by default. But these columns are often removed by crontab editing software or cron expert users who don't feel the need to keep them. Finally, since most people edit crontab files with simple editors, no validation is performed on the actual script that they request is run. For example, is your script in /usr/bin/ or /usr/local/bin? Did you remember to check? The editor sure didn't check for you.

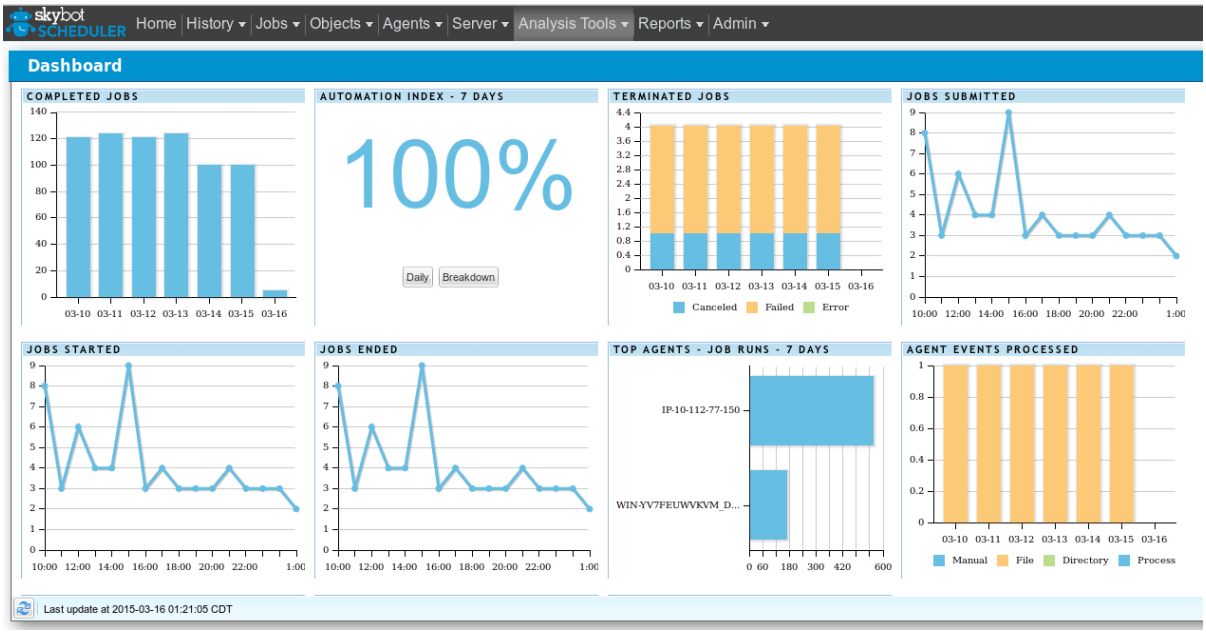


FIGURE 1. **System administrators need to know which jobs ran and which jobs failed. Skybot by HelpSystems brings all of this information to you on a simple to use dashboard.**

These are all simple, "only human" mistakes. If you use cron long enough, you learn to be careful and double-check any changes you make. Usually this lesson is learned the hard way though. And if you share system administration duties with others, you have to wonder whether the other system administrators are exercising the same diligence that you are.

Wouldn't it be nice if the software you used to schedule jobs guided you a bit by presenting viable options and verifying input in order to prevent mistakes? Wouldn't it be nice to be able to use your mouse to click on buttons to schedule a job? Instead of writing:

```
0   23   *   *   mon,tue,wed,thu,fri   /usr/local/bin/do_backups.sh
```

Software never will be a replacement for human intelligence, but it should at least watch your back.

you simply could click on "at midnight, on the hour, Monday through Friday, run the backup job". Then the software would verify that the job you scheduled actually exists and is able to run. Software never will be a replacement for human intelligence, but it should at least watch your back. By the way, did you notice the typo in that crontab entry?

Multi-Server-Friendly

Back in the good-old days, an IT enterprise might have consisted of an e-mail server and a file server or two, or maybe a handful of each. Usually, the system administrator would use one of those servers as a print server as well. Times have changed. Today's IT enterprise often adds Web servers, authentication servers, database servers, (sometimes many different databases too) calendaring servers, business process management servers, CMS servers, revision control servers, and backup servers. In today's enterprise, often there are servers to monitor the other servers!

In addition to running an enterprise with more servers that offer more and more different services, today's enterprise may employ virtualization to help drive costs down. Virtualization creates situations where servers can

come into existence and disappear based on demand requirements. Even a virtual server may require that certain jobs run on schedule. And since virtual machines don't require an increase in physical chassis count, there is a propensity simply to spin up a new virtual machine any time a customer or department manager requests a new server. This results in a higher "virtual chassis" count. But, even a virtual server has some job scheduling requirements. Logs need to be rotated periodically, and depending on how you architect your enterprise, it might make sense to run backups from within the virtual machine or container.

So modern system administrators are faced with a two-fold problem: they have to manage a wider variety of servers, and they have to manage a larger number of them.

From a technical point of view, it's not too terribly difficult to replicate a crontab file to any given number of servers. It's just a matter of a bit of scripting. But from a practical point of view, it's just another manual step in what should be an automated process.

But, then it gets worse. Not every server needs the same crontab file. A Web server requires different jobs to run from what a database server would. And the accounting department may have jobs that need to run on its servers before jobs in payroll can run. However, all the servers in a given organization may share a core set of jobs that run across the entire enterprise. Managing this could prove painful.

In an ideal world, you'd like to be able to categorize your servers based on things like operating system,

function, department, or development cycle such as development or production.

You'd like to be able to run a log analysis job on your Web servers that wouldn't be meaningful to run on your database servers, even though the database servers feed their data to the Web servers. On the other hand, your database servers require periodic table maintenance that the mail servers don't require.

If your sales department servers are customer-facing, you might decide to run their server backups outside business hours and run the accounting department's backups during the day so that their processing jobs can run the following night.

Certainly, your Linux servers require different scheduled jobs to run than your BSD and Windows servers, but you'd still like to manage them all from a central location.

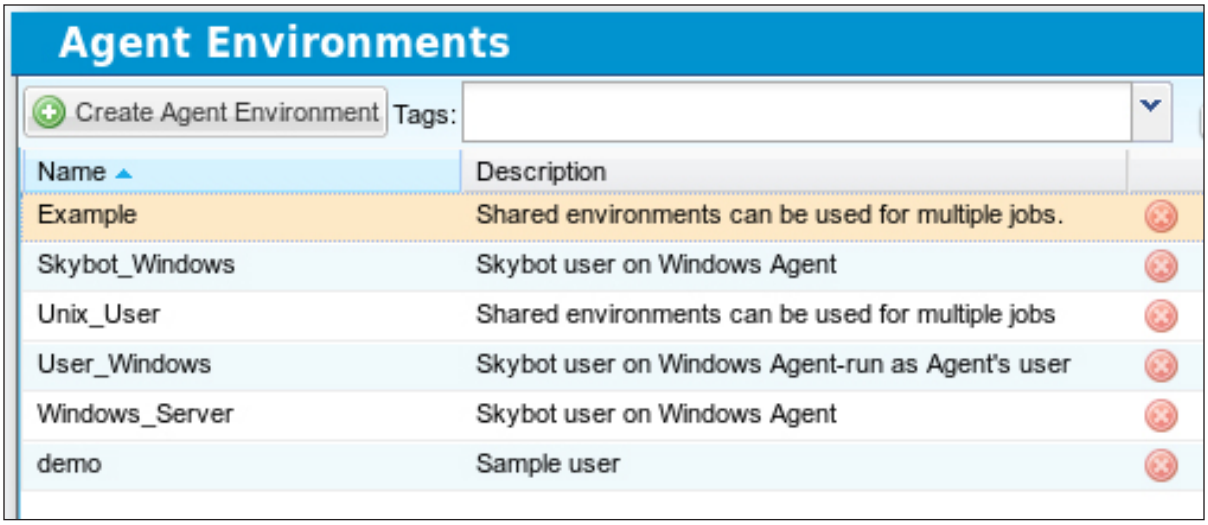


FIGURE 2. The ideal job scheduling tool should work the same on Windows and various flavors of UNIX.

One area of job scheduling that cron just doesn't do well at is job dependency management. Some jobs simply can not run until other jobs have completed.

Finally, you might like to back up your development servers more frequently than the corresponding production servers, because the development servers change more frequently.

Being able to categorize servers and schedule jobs on each server according to various criteria enables system administrators to manage a larger enterprise more efficiently. A good software tool would make managing these criteria and the distribution of job schedules as efficient and effortless as possible.

Dependency Management

One area of job scheduling that cron just doesn't do well is job dependency management. Some jobs simply cannot run until other jobs have completed.

A classic example is the database server that requires scheduled maintenance each night. It doesn't do any good to run the backup job on a database server that is still running its daily maintenance. In fact, it can be devastating. At best, both jobs conflict with each other and slow each other down, causing them to run longer

than needed. At worst, one job corrupts the other; the backup job backs up a database that is only partially repaired, or the repair job attempts to repair a database that is locked by the backup job.

As another example, it might make sense to process a company's accounts payable (AP) before processing the accounts receivable (AR). In this way, you can be sure that your accounts never are negative since all of the outgoing money was spent before any of the incoming money came in. Of course, if you work at a bank, it might make sense to process withdrawals before deposits so as to increase the potential of collecting insufficient funds penalties.

And to put this all together, it might make sense to run the database backup job after the maintenance job, which should be run only once *both* the AP and AR jobs have completed.

Typically, this type of goal is accomplished by clever, and perhaps sloppy, job scheduling. You would schedule the AP processing right at 8 p.m. on the East coast, which is still 5 p.m. on the West coast. Then, knowing that the AP job typically only takes an hour to run, you would consider running the AR job at 10 p.m., giving yourself an hour of leeway in case the AP job takes longer than normal. Using the same logic, you schedule the database repair at 11 p.m., and finally, the backup at 1 a.m. the next morning. Remember, this is only about four hours of processing. But because of the limitations in job scheduling with cron, you've had to string it out over eight hours. Just in case there's a problem with any of these jobs, a system

administrator would need to be on call while they run, and there's a big difference between being on call until 9 p.m. and being on call until 1 a.m.!

Simply put, cron doesn't understand job dependency, so you have to resort to clever, and inefficient, job sequencing.

The situation gets even worse when the various jobs that depend upon each other are run on different servers, perhaps in different data centers spread out across timezones.

It would be nice to be able to schedule a job to run as soon as all of its dependencies have been met. In this case, if a given job runs faster than normal, the other jobs get to run sooner. On the other hand, if a job runs longer than normal, it just runs longer, and the other jobs simply wait until it's time for them to run.

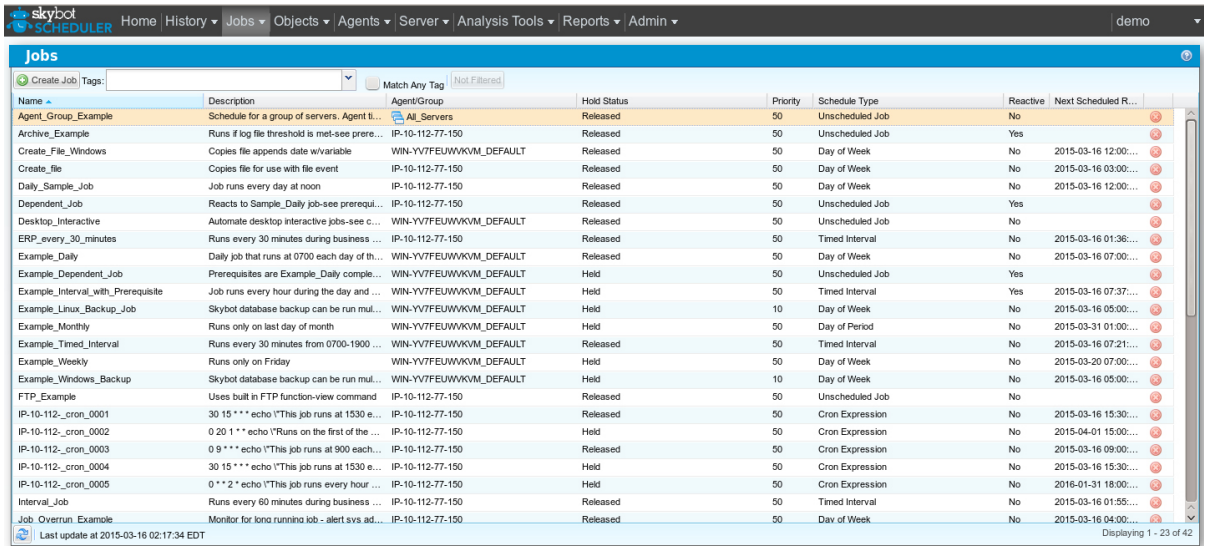


FIGURE 3. The status of every job on every server is just a few clicks away.

This kind of dependency management could be accomplished using a make file, or another tool of that nature. But even that doesn't solve the problem of being able to schedule jobs only after *each* and *every* server involved has run its backups and had its logs rotated. Dependency management is one thing; dependency management across multiple servers is quite another, and it's really hard to accomplish with "clever scheduling" and make files.

A good job scheduling tool needs to understand job dependencies and understand that some of those dependencies may not be on the same server.

Easy to Visualize

So, over time, you've managed to build up a large array of scheduled jobs that automate much of the day-to-day operations of your enterprise. You've created a complex hierarchy of jobs and the servers that they run on. Everything is working perfectly, and you are a happy camper. Then it happens. Someone asks you how it all works.

Usually, this someone is another system administrator or maybe a new, or junior, system administrator. Or, in the worst case, it's your manager, and he or she wants to understand fully how it all "fits together". Printing out the crontab files for all of your servers obviously isn't going to provide anyone with the big picture. You might be able to explain when the various jobs on a given server run, but that doesn't even begin to explain *why* they run when they do. Perhaps you have some job dependency

Trying to make sense of the crontab files from multiple servers could be like drinking from a fire hydrant.

issues, like I discussed earlier, and you've had to schedule jobs with that in mind. Maybe you scheduled them to run when they run because it seemed like a good idea at the time. To put it differently, is job A scheduled after job B because it's convenient, or because job B actually depends on job A completing first? Just looking at when a job is scheduled doesn't capture that distinction.

Trying to make sense of the crontab files from multiple servers could be like drinking from a fire hydrant. Most parts of the files might even be identical, in which case, the devil is in the subtle differences between the various files.

You also might find yourself in the situation where the Web development team members want to understand how their stuff works, but they really don't care how the database administrators' stuff works. To put a perverse twist on things, the DBAs may ask to have their backup jobs scheduled earlier in the day, but only because they don't understand that their backups can't run until the accounting department completes its processing.

Nobody understands how it all works but you, and you don't want to spend the rest of your life in meetings trying to explain it all.

An enterprise-grade scheduling tool should help you visualize when jobs run and what they depend upon before they can run. This visualization should be presented in a form that is intuitive and approachable by people with a wide range of technical sophistication. Moreover, this visualization should be based on how things *actually* work, not just on how they are documented to work. Maintaining the documentation on how things work shouldn't be a separate task that needs to be (and almost never is) done when changes are made.

Finally, an enterprise-grade scheduling tool should be able to understand the entire enterprise, not just one or a few servers. The tool should be able to correlate how a job that runs on one server affects jobs that run on any number of other servers. And rather than giving everybody the same big picture, it would be nice if the scheduling tool could show stakeholders just what they need to see and nothing else.

Delegation of Authority

Most system administrators are extremely busy. Scheduling, documenting, and monitoring jobs is just simply something that needs to be delegated to other staff, if available. Perhaps the Web developers want to be able to manage the jobs that run on their servers, while the DBAs certainly don't want Web developers breaking things on their servers. Although the tier 1 support staff needs to be able to see what jobs are

scheduled and determine if they ran correctly, they may not need the ability to change job schedules. Managers are always curious, and rightly so, but no one wants them doing anything other than looking at what's scheduled and where.

A lot of people have an interest in enterprise job scheduling. Some of them need only to be able to look, and others obviously need to be able to make changes to a given job's schedule. These roles may reflect the relative skill level of the various staff members, such as the difference between a tier 1 technician and a senior system administrator. Or, the roles may be segmented by departmental boundaries; the development department doesn't need to understand how the sales department operates.

There's nothing wrong with compartmentalizing operations like this. Not only does it keep people from inadvertently making changes that affect others, but it also makes it easier for staff members to understand how their department operates, since their understanding isn't clouded by having to understand complex interactions between departments.

Making sure that changes are being made only by those people who are authorized and qualified to change job schedules is just part of the equation. The other side of the equation is knowing who made what changes and when. Change logging and accountability is important. If something is found to be broken, it's often helpful to be able to know what was changed. Also, if changes are made incorrectly, this might point out an opportunity for

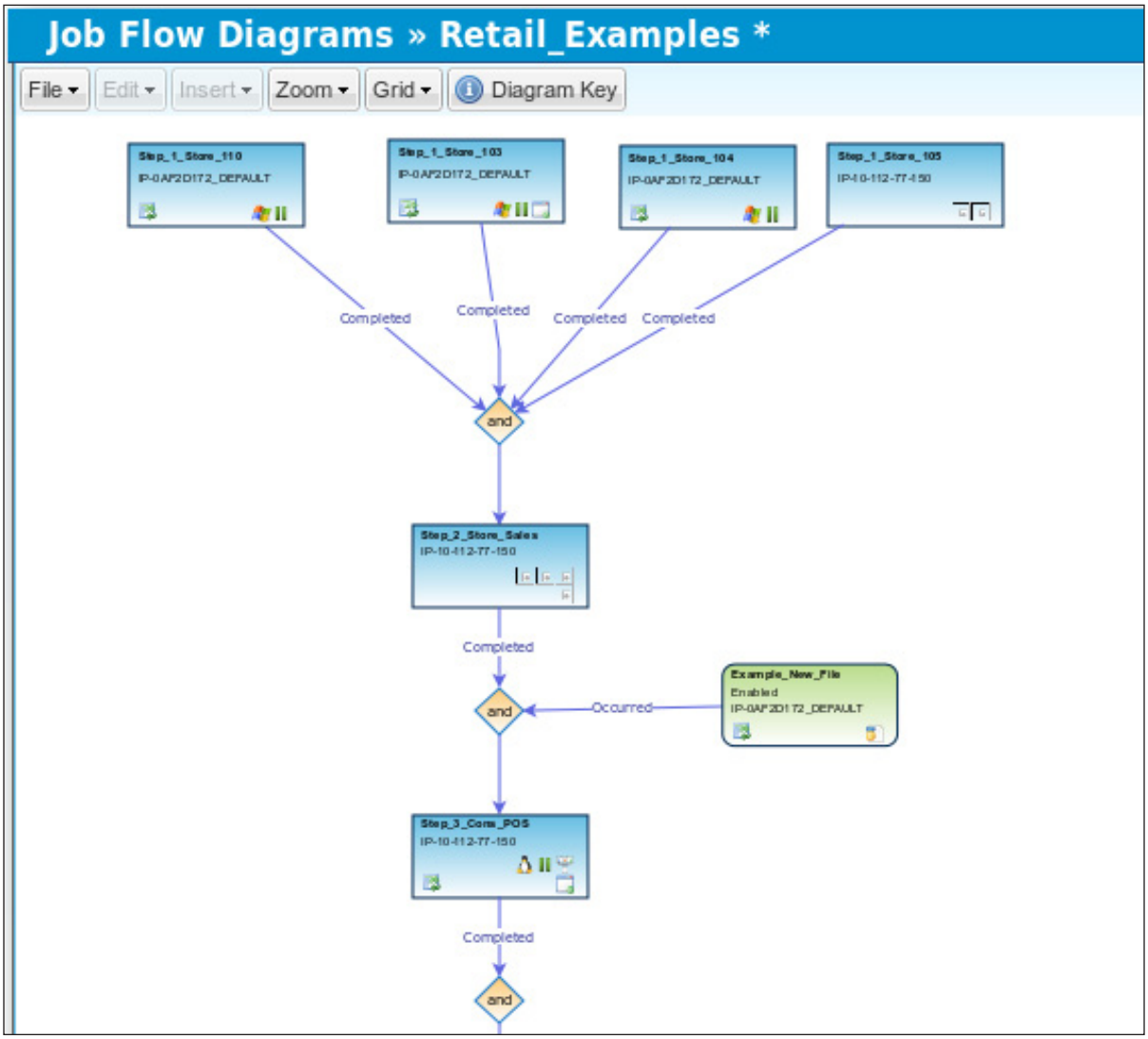


FIGURE 4. **Job flow diagrams make process dependencies easy to understand. Role-based diagrams provide different stakeholders with the information they need.**

additional training for the staff member or department head who made the change. The purpose here isn't to start a witch hunt. The purpose is just to make sure that the people who need to know about changes always know about changes.

Being able to delegate the ability to make changes, and to keep people accountable for the changes they make, actually saves time. Changes don't have to be escalated to more senior staff if they can be handled by other staff members. Managers who just need to know how things work can be put into self-serve mode without having to take up the time of busy support staff. And finally, if a change is made that breaks something, logging is the best way to find out what was done and how to undo it.

Management by Exception

Most system administrators are used to checking their e-mail in the morning and seeing a number of messages triggered by the various jobs that ran the night before. Usually, perusing these messages becomes part of their morning workflow. At first, you actually read every line of every e-mail message. Over time, you get used to not finding anything untoward in these messages, so you simply begin to skim over them, looking for log entries that "stick out" at you. Eventually, you are tempted to become complacent and simply ignore the messages. This is the normal progression and is quite common—until it finally happens. One of the status reports indicates something went wrong. Usually this indication is camouflaged in the middle of the e-mail message, neatly tucked away between two perfectly normal log entries.

In the worst-case scenario, this error indication goes unnoticed for a few days. By then, it usually becomes a crisis situation. A hard drive that *was* merely failing a few days ago, suddenly dies today, for example. A log entry that might

A job scheduling tool that is worthy of running the entire enterprise should be smart enough to tell the administrators only when something goes wrong instead of drowning them in a flood of good, but unimportant, news.

have indicated that your Web site had been compromised becomes a situation where your Web site is used to send SPAM, and then your mail server gets blacklisted, breaking e-mail for your entire enterprise.

Logs are important. But no one actually has time to read them, unless they're important, and you can't tell if they're important unless you read them—all of them.

As odd as it sounds, it would be so much easier if system administrators received only the bad news. This is where "management by exception" comes into play. System administrators never need to react to having things run as expected. This is normal. System administrators like this. The only time system administrators should need to put down their coffee cups is when something is broken. This is the exception, and it should be indicated unambiguously by a clear error indication.

A job scheduling tool that is worthy of running the entire enterprise should be smart enough to tell the administrators only when something goes wrong instead of drowning them in a flood of good, but unimportant, news.

Flexible Scheduling

The cron scheduling system is pretty good at scheduling simple recurring jobs. It's easy to schedule a job to run at midnight every night. Scheduling a job to run every 15 minutes, but only during the business week, is just a little bit more difficult, but it's not something you'd spend more than ten minutes doing. Let's face it, cron is pretty good at what it does and is pretty easy to use, but its scheduling capabilities are limited. For example, sometimes you actually want to run a job on the very last day of the month, instead of merely running it very early on the first day of the month.

Enterprise job scheduling isn't just about making sure jobs run at a particular time. Sometimes you want a job to run based on an external trigger—for example, a network intrusion detection system might create a log file in response to suspicious activity being detected. Although many network intrusion detection systems can run scripts in response to various triggers, it might make sense to consolidate that type of job under the control of the scheduling system.

On the other hand, you might want a job to run when a particular host is no longer reachable via ICMP. Once again, your network management tools may be able to perform the same function, but an enterprise job scheduler may be able to do it in a more flexible fashion.

This is just a matter of being able to use the tool that does the job best, and it seems reasonable to expect a job scheduling tool to be able to manage various jobs no matter what triggered them to run.

Revision Control

Most programmers are familiar with using a version control system to track changes that are made to one or more files, but most system administrators aren't in the habit of putting system configuration files under version control. For one thing, it's yet another tool that has to be set up and maintained. Then there is the additional staff training. Finally, everyone has to remember to use it.

Revision control allows system administrators to examine and potentially back out changes that were made to a file, even if the changes were made long ago. Revision control doesn't rely on your backups and even can bring back versions of a given file that have long since expired in your backup regimen. Unfortunately, cron isn't version-control-friendly.

Everyone has been there. Someone made changes to a file two weeks ago, but nobody remembers what changes were made. Of course, backups were kept only for a week, so there's no going to the backups to see what changed. If the file changes had been made under version control, you'd be able to see exactly what changes were made and even back them out if needed.

Revision control isn't something normally applied to job scheduling, but it would sure be nice to have.

Conclusion

Cron is a great tool that's been around for a long time. As a system administrator in today's world, however, you are managing more servers, real and virtual, than ever before, and you need to automate any process you can. In order to make daily operations as efficient as possible, you need a tool that

FIGURE 5. **Visit http://www.helpsystems.com/skybot to get started with Skybot.**

allows you to manage jobs on any number of different servers easily. You also likely now find yourself working in a larger and more diverse group of administrators, managers, and developers than ever before, so you need a means of not only documenting how the enterprise operates, but also of delegating authority over how it operates. Modern enterprise job scheduling tools, such as Skybot by HelpSystems, offer superior functionality, ease of use, and a more modern, graphical user interface. Visit the Skybot Web site for more details and to try out the on-line demonstration: http://www.helpsystems.com/skybot.■