SysAdmin 101

By Kyle Rankin

LINUX JOURNAL

Table of Contents

SYS ADMIN 101 By Kyle Rankin

About the Author

Kyle Rankin is a Tech Editor and columnist at Linux Journal and the Chief Security Officer at Purism. He is the author of *Linux Hardening in Hostile Networks*, *DevOps Troubleshooting*, *The Official Ubuntu Server Book*, *Knoppix Hacks*, *Knoppix Pocket Reference*, *Linux Multimedia Hacks* and *Ubuntu Hacks*, and also a contributor to a number of other O'Reilly books. Rankin speaks frequently on security and open-source software including at BsidesLV, O'Reilly Security Conference, OSCON, SCALE, CactusCon, Linux World Expo and Penguicon. You can follow him at @kylerankin.

Introduction

KYLE RANKIN

This book explores system administrator fundamentals. These days, DevOps has made even the job title "system administrator" seem a bit archaic, much like the "systems analyst" title it replaced. These DevOps positions are rather different from typical sysadmin jobs in the past in that they have a much larger emphasis on software development far beyond basic shell scripting. As a result, they often are filled with people with software development backgrounds without much prior sysadmin experience. In the past, sysadmins would enter the role at a junior level and be mentored by a senior sysadmin on the team, but in many cases currently, companies go quite a while with cloud outsourcing before their first DevOps hire. As a result, DevOps engineers might be thrust into the role at a junior level with no mentor around apart from search engines and Stack Overflow posts. In this book, I expound on some of the lessons I've learned through the years that might be obvious to longtime sysadmins but may be news to someone just coming into this position.

Alerting

In this first chapter, I cover on-call alerting. Like with any job title, the responsibilities given to sysadmins, DevOps and Site Reliability Engineers may differ, and in some cases, they may not involve any kind of 24x7 on-call duties, if you're lucky. For everyone else, though, there are many ways to organize on-call alerting, and there also are many ways to shoot yourself in the foot.

The main enemies of on-call alerting are false positives, with the main risks being ignoring alerts or burnout for members of your team. This chapter describes some best practices you can apply to your alerting policies that hopefully will reduce burnout and make sure alerts aren't ignored.

Alert Thresholds

A common pitfall sysadmins run into when setting up monitoring systems is to alert on too many things. These days, it's simple to monitor just about any aspect of a server's health, so it's tempting to overload your monitoring system with all kinds of system checks. One of the main ongoing maintenance tasks for any monitoring system is setting appropriate alert thresholds to reduce false positives. This means the more checks you have in place, the higher the maintenance burden. As a result, I have a few different rules I apply to my monitoring checks when determining thresholds for notifications.

Critical alerts must be something I want to be woken up about at 3am.

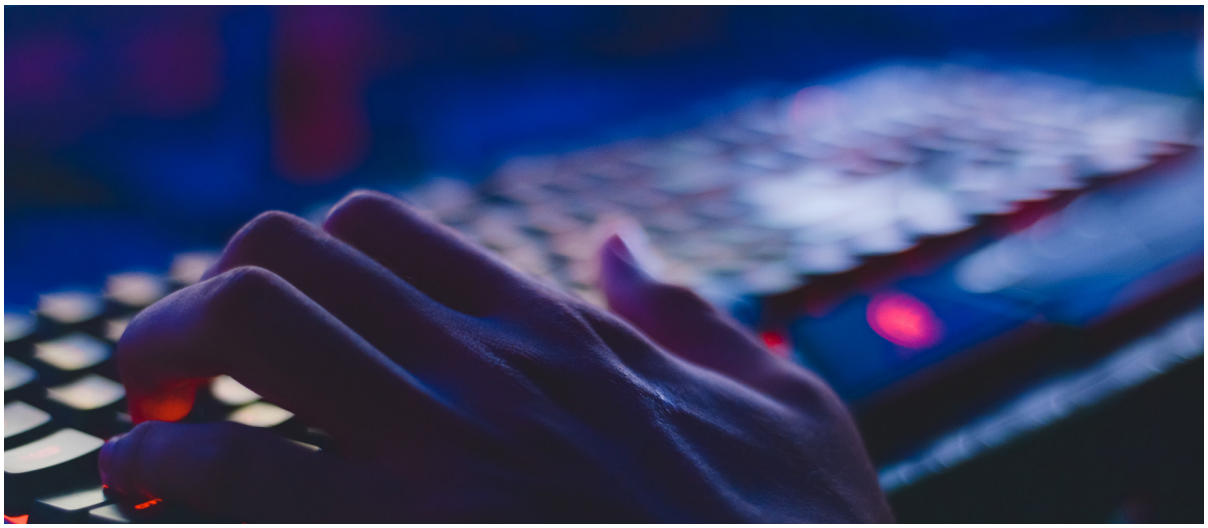
A typical cause of sysadmin burnout is being woken up with alerts for systems that don't matter. If you don't have a 24x7 international development team, you probably don't care if the build server has a problem at 3am, or even if you do, you probably are going to wait until the morning to fix it. By restricting critical alerts to just those systems that must be online 24x7, you help reduce false positives and make sure that real problems are addressed quickly.

Critical alerts must be actionable.

Some organizations send alerts when just about anything happens on a system. If I'm being woken up at 3am, I want to have a specific action plan associated with that alert so I can fix it. Again, too many false positives will burn out a sysadmin that's on call, and nothing is more frustrating than getting woken up with an alert that you can't do anything about. Every critical alert should have an obvious action plan the sysadmin can follow to fix it.

Warning alerts tell me about problems that will be critical if I don't fix them.

There are many problems on a system that I may want to know about and may want to investigate, but they aren't worth getting out of bed at 3am. Warning alerts don't trigger a pager, but they still send me a quieter notification. For instance, if load,



used disk space or RAM grows to a certain point where the system is still healthy but if left unchecked may not be, I get a warning alert so I can investigate when I get a chance. On the other hand, if I got only a warning alert, but the system was no longer responding, that's an indication I may need to change my alert thresholds.

Repeat warning alerts periodically.

I think of warning alerts like this thing nagging at you to look at it and fix it during the work day. If you send warning alerts too frequently, they just spam your inbox and are ignored, so I've found that spacing them out to alert every hour or so is enough to remind me of the problem but not so frequent that I ignore it completely.

Everything else is monitored, but doesn't send an alert.

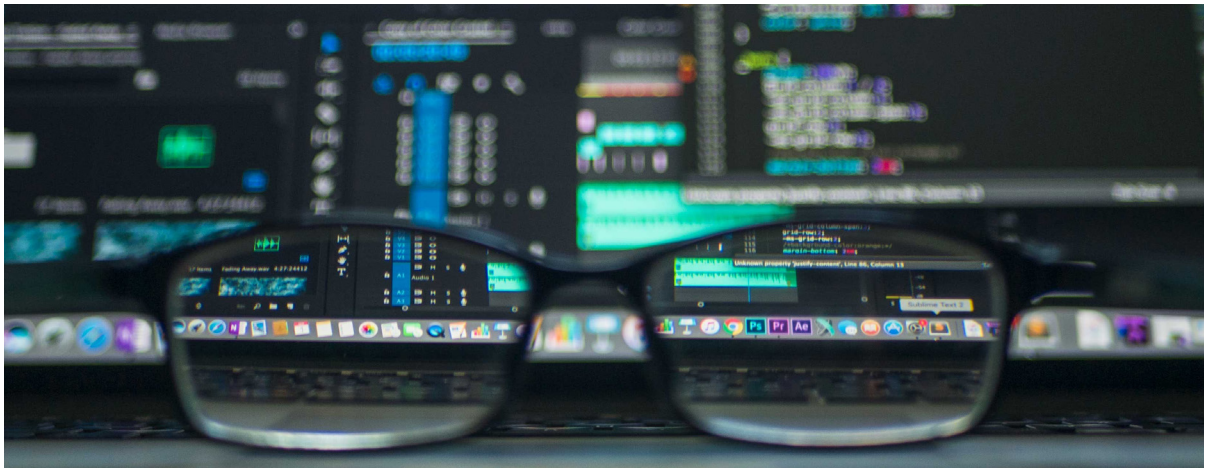
There are many things in my monitoring system that help provide overall context when I'm investigating a problem, but by themselves, they aren't actionable and aren't anything I want to get alerts about. In other cases, I want to collect metrics from my systems to build trending graphs later. I disable alerts altogether on those kinds of checks. They still show up in my monitoring system and provide a good audit trail when I'm investigating a problem, but they don't page me with useless notifications.

Kyle's rule.

One final note about alert thresholds: I've developed a practice in my years as a sysadmin that I've found is important enough as a way to reduce burnout that I take it with me to every team I'm on. My rule is this:

> If sysadmins were kept up during the night because of false alarms, they can clear their projects for the next day and spend time tuning alert thresholds so it doesn't happen again.

There is nothing worse than being kept up all night because of false positive alerts and knowing that the next night will be the same and that there's nothing you can do about it. If that kind of thing continues, it inevitably will lead either to burnout or to sysadmins silencing their pagers. Setting aside time for sysadmins to fix false alarms helps, because they get a chance to improve their night's sleep the next night. As a team lead or manager, sometimes this has meant that I've taken on



a sysadmin's tickets for them during the day so they can fix alerts.

Paging

Sending an alert often is referred to as paging or being paged, because in the past, sysadmins, like doctors, carried pagers on them. Their monitoring systems were set to send a basic numerical alert to the pager when there was a problem, so that sysadmins could be alerted even when they weren't at a computer or when they were asleep. Although we still refer to it as paging, and some older-school teams still pass around an actual pager, these days, notifications more often are handled by alerts to mobile phones.

The first question you need to answer when you set up alerting is what method you will use for notifications. When you are deciding how to set up pager notifications, look for a few specific qualities.

Something that will alert you wherever you are geographically.

A number of cool office projects on the web exist where a broken software build triggers a big red flashing light in the office. That kind of notification is fine for office-hour alerts for non-critical systems, but it isn't appropriate as a pager notification even during the day, because a sysadmin who is in a meeting room or at lunch would not be notified. These days, this generally means some kind of notification needs to be sent to your phone.

An alert should stand out from other notifications.

False alarms can be a big problem with paging systems, as sysadmins naturally will start ignoring alerts. Likewise, if you use the same ringtone for alerts that you use for any other email, your brain will start to tune alerts out. If you use email for alerts, use filtering rules so that on-call alerts generate a completely different and louder ringtone from regular emails and vibrate the phone as well, so you can be notified even if you silence your phone or are in a loud room. In the past, when BlackBerries were popular, you could set rules such that certain emails generated a "Level One" alert that was different from regular email notifications.

The BlackBerry days are gone now, and currently, many organizations (in particular startups) use Google Apps for their corporate email. The Gmail Android application lets you set per-folder (called labels) notification rules so you can create a filter that moves all on-call alerts to a particular folder and then set that folder so that it generates a unique alert, vibrates and does so for every new email to that folder. If you don't

have that option, most email software that supports multiple accounts will let you set different notifications for each account so you may need to resort to a separate email account just for alerts.

Something that will wake you up all hours of the night.

Some sysadmins are deep sleepers, and whatever notification system you choose needs to be something that will wake them up in the middle of the night. After all, servers always seem to misbehave at around 3am. Pick a ringtone that is loud, possibly obnoxious if necessary, and also make sure to enable phone vibrations. Also configure your alert system to re-send notifications if an alert isn't acknowledged within a couple minutes. Sometimes the first alert isn't enough to wake people up completely, but it might move them from deep sleep to a lighter sleep so the follow-up alert will wake them up.

While ChatOps (using chat as a method of getting notifications and performing administration tasks) might be okay for general non-critical daytime notifications, they are not appropriate for pager alerts. Even if you have an application on your phone set to notify you about unread messages in chat, many chat applications default to a "quiet time" in the middle of the night. If you disable that, you risk being paged in the middle of the night just because someone sent you a message. Also, many third-party ChatOps systems aren't necessarily known for their mission-critical reliability and have had outages that have spanned many hours. You don't want your critical alerts to rely on an unreliable system.

Your notification system needs to be reliable and able to alert you quickly at all times.

Something that is fast and reliable.

Your notification system needs to be reliable and able to alert you quickly at all times. To me, this means alerting is done in-house, but many organizations opt for third parties to receive and escalate their notifications. Every additional layer you can add to your alerting is another layer of latency and another place where a notification may be dropped. Just make sure whatever method you choose is reliable and that you have some way of discovering when your monitoring system itself is offline.

In the next section, I cover how to set up escalations—meaning, how you alert other members of the team if the person on call isn't responding. Part of setting up escalations is picking a secondary, backup method of notification that relies on a different infrastructure from your primary one. So if you use your corporate Exchange server for primary notifications, you might select a personal Gmail account as a secondary. If you have a Google Apps account as your primary notification, you may pick SMS as your secondary alert.

Email servers have outages like anything else, and the goal here is to make sure that even if your primary method of notifications has an outage, you have some alternate way of

finding out about it. I've had a number of occasions where my SMS secondary alert came in before my primary just due to latency with email syncing to my phone.

Create some means of alerting the whole team.

In addition to having individual alerting rules that will page someone who is on call, it's useful to have some way of paging an entire team in the event of an "all hands on deck" crisis. This may be a particular email alias or a particular key word in an email subject. However you set it up, it's important that everyone knows that this is a "pull in case of fire" notification and shouldn't be abused with non-critical messages.

Alert Escalations

Once you have alerts set up, the next step is to configure alert escalations. Even the best-designed notification system alerting the most well intentioned sysadmin will fail from time to time either because a sysadmin's phone crashed, had no cell signal, or for whatever reason, the sysadmin didn't notice the alert. When that happens, you want to make sure that others on the team (and the on-call person's second notification) is alerted so someone can address the alert.

Alert escalations are one of those areas that some monitoring systems do better than others. Although the configuration can be challenging compared to other systems, I've found Nagios to provide a rich set of escalation schedules. Other organizations may opt to use a third-party notification system specifically because their chosen monitoring solution

doesn't have the ability to define strong escalation paths. A simple escalation system might look like the following:

- Initial alert goes to the on-call sysadmin and repeats every five minutes.
- If the on-call sysadmin doesn't acknowledge or fix the alert within 15 minutes, it escalates to the secondary alert and also to the rest of the team.
- These alerts repeat every five minutes until they are acknowledged or fixed.

The idea here is to give the on-call sysadmin time to address the alert so you aren't waking everyone up at 3am, yet also provide the rest of the team with a way to find out about the alert if the first sysadmin can't fix it in time or is unavailable. Depending on your particular SLAs, you may want to shorten or lengthen these time periods between escalations or make them more sophisticated with the addition of an on-call backup who is alerted before the full team. In general, organize your escalations so they strike the right balance between giving the on-call person a chance to respond before paging the entire team, yet not letting too much time pass in the event of an outage in case the person on call can't respond.

If you are part of a larger international team, you even may be able to set up escalations that follow the sun. In that case, you would select on-call administrators for each geographic region and set up the alerts so that they were aware of the different time periods and time of day in those regions, and

then alert the appropriate on-call sysadmin first. Then you can have escalations page the rest of the team, regardless of geography, in the event that an alert isn't solved.

On-Call Rotation

During World War One, the horrors of being in the trenches at the front lines were such that they caused a new range of psychological problems (labeled shell shock) that, given time, affected even the most hardened soldiers. The steady barrage of explosions, gun fire, sleep deprivation and fear day in and out took its toll, and eventually both sides in the war realized the importance of rotating troops away from the front line to recuperate.

It's not fair to compare being on call with the horrors of war, but that said, it also takes a kind of psychological toll that if left unchecked, it will burn out your team. The responsibility of being on call is a burden even if you aren't alerted during a particular period. It usually means you must carry your laptop with you at all times, and in some organizations, it may affect whether you can go to the movies or on vacation. In some badly run organizations, being on call means a nightmare of alerts where you can expect to have a ruined weekend of firefighting

every time. Because being on call can be stressful, in particular if you get a lot of nighttime alerts, it's important to rotate out sysadmins on call so they get a break.

The length of time for being on call will vary depending on the size of your team and how much of a burden being on call is. Generally speaking, a one- to four-week rotation is common, with two-week rotations often hitting the sweet spot. With a large enough team, a two-week rotation is short enough that any individual member of the team doesn't shoulder too much of the burden. But, even if you have only a three-person team, it means a sysadmin gets a full month without worrying about being on call.

Holiday on call.

Holidays place a particular challenge on your on-call rotation, because it ends up being unfair for whichever sysadmin it lands on. In particular, being on call in late December can disrupt all kinds of family time. If you have a professional, trustworthy team with good teamwork, what I've found works well is to share the on-call burden across the team during specific known holiday days, such as Thanksgiving, Christmas Eve, Christmas and New Year's Eve. In this model, alerts go out to every member of the team, and everyone responds to the alert and to each other based on their availability. After all, not everyone eats Thanksgiving dinner at the same time, so if one person is sitting down to eat, but another person has two more hours before dinner, when the alert goes out, the first person can reply "at dinner", but the next person can reply "on it", and that way, the burden is shared.

If you are new to on-call alerting, I hope you have found this list of practices useful. You will find a lot of these practices in place in many larger organizations with seasoned sysadmins, because over time, everyone runs into the same kinds of problems with monitoring and alerting. Most of these policies should apply whether you are in a large organization or a small one, and even if you are the only DevOps engineer on staff, all that means is that you have an advantage at creating an alerting policy that will avoid some common pitfalls and overall burnout.

Automation

In the first chapter of this ebook, I explore how to approach on-call rotations as a sysadmin. In this chapter, I discuss how to automate yourself out of your job. There is a quote that you see from time to time in sysadmin circles that goes something along the lines of "Be careful or I will replace you with a tiny shell script." Good system administrators hate performing mundane tasks and constantly seek to apply that saying to themselves. That said, there are many different approaches to automation, and not all of them result in a time-savings. Here, I discuss my experience with automation and describe what, when, why and how you should (and shouldn't) automate.

Why You Should Automate

There are a number of different reasons why you should take steps to automate your work as a sysadmin:

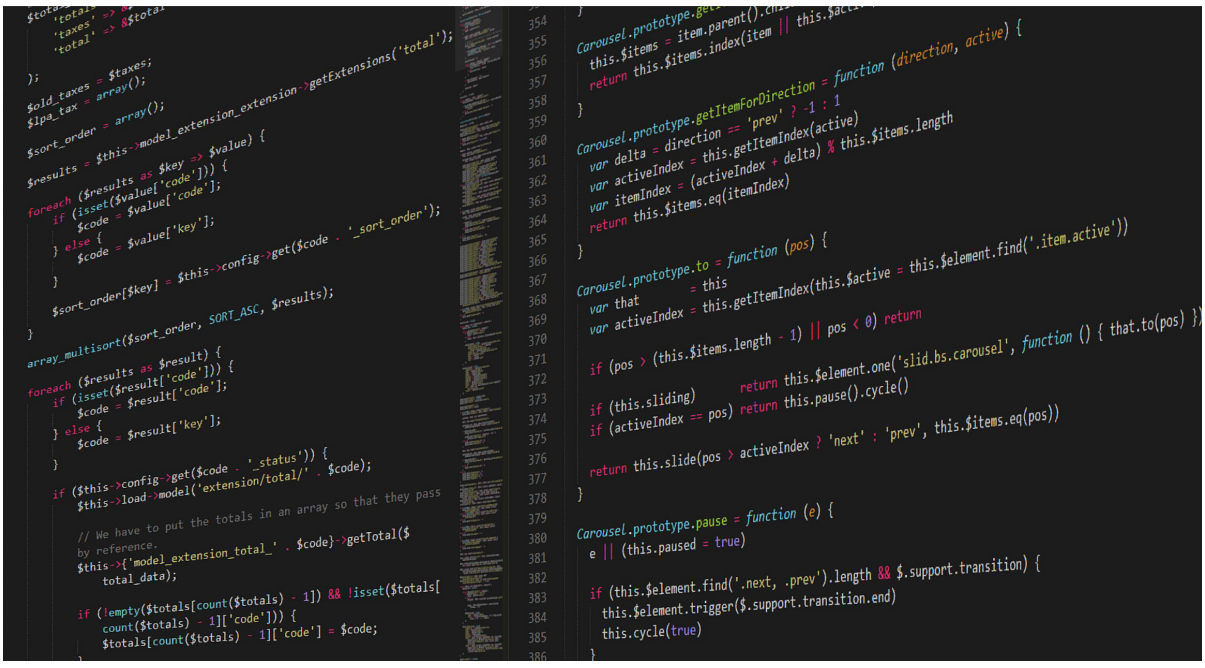
1) It frees up time spent doing mundane tasks to focus on more important work.

With all of the automation that's already built in to servers these days, it's easy to take for granted just how many mundane tasks sysadmins have had to perform in the past. Logs weren't always rotated automatically; backups usually were home-grown affairs that often were triggered manually. Even

now, there still are system administrators who install every single server by hand, log in to a machine manually and install or update software, and configure server configuration files on the host by hand.

Let's take server OS installation as an example—a modern interactive server OS installation may take anywhere from 15 minutes to an hour of sysadmin time to walk through and answer questions. These are the kinds of actions that don't really require a sysadmin's expertise once you've made the initial decisions about how you want a server to be set up. By automating these mundane tasks, you can get back to the more difficult work that does require your expertise.

2) Automation reduces mistakes in routine tasks.



The thing about performing the same task over and over by hand is that it is easy to make mistakes, and if it's something you do every day, eventually you even may stop paying attention to whether your task succeeded. Also, the way that you may perform a certain task might be a little bit different from how a different administrator on the team does it. By automating a task, the team can agree on the ideal way to perform it and know that when you run your automation script, it is performed the same way every single time with no skipped steps or commands run in the wrong order.

3) Automation allows everyone on the team to be productive. With automation, you can take even a complex process and reduce it down to a command. That command then becomes something that anyone on the team can run, whereas the complex process may have required more senior members of the team. For instance, if you take production software deployment as an example, often there can be a complex arrangement of triggering load balancer and monitoring maintenance modes, software versions to check, mirrors to sync up, and services to restart and test. Even though these individual steps may be mundane, combined, they become pretty complicated and could overwhelm a junior member of the team—especially when production uptime hangs in the balance. By automating that process, senior administrators can put all of their expertise into creating the right process that performs the right checks, and they can go on vacation knowing that anyone else on the team now can perform the task the right way.

4) Automation reduces documentation workload.

Often instead of automating a task, a sysadmin team will spend time documenting a process. There is still an important place for documentation, and in the next section, I discuss when that makes sense and when it doesn't. The fact is though, if you take take an entire process and put it into a single automated task, you no longer need a full wiki page of documentation (that inevitably will become out of date), because you've reduced it down to "run this command". Because the process is now automated, you also know the process is kept up to date; otherwise, the script wouldn't work.

What You Should Automate

Not everything is appropriate for automation, and even things that may be good candidates for automation may not be good candidates today (the next section covers when you should

automate). Following are a few different types of tasks that make good candidates for automation.

1) Routine tasks.

In general, tasks that you perform frequently (at least monthly) are good candidates for automation. The more frequent the task, in theory, the more time-savings you would get from automating it. Tasks that you perform only once a year may not be worth the effort to build automation around, and instead, those are the kinds of tasks that benefit from good documentation.

2) Repeatable tasks.

If you could document a process as a series of commands, and then copy and paste them one by one in a terminal and the task would be complete, that's a repeatable task that may be a good candidate for automation. On the other hand, one-off tasks that have custom inputs or are something you may never have to do again aren't worth the time and effort to automate.

3) Complex tasks.

The more complex a task, the more opportunities you have for mistakes if you do it manually. If a task has multiple steps, in particular steps that require you to take the output from one step and use it as input for another, or steps that use commands with a complex string of arguments, these are all great candidates for automation.

4) Time-consuming tasks.

The longer the tasks take to complete (especially if there are

periods of running a command, waiting for it to complete, and then doing something with that command's output), the better a candidate it is for automation. OS installation and configuration is a great example of this, as when you install an OS, there are periods when you enter installation settings and periods when you wait for the installation to complete. All of that waiting is wasted time. By automating long-running tasks, you can go do some other work and come back to the automation (or better, have it alert you) to see if it is complete.

When You Should Automate

My coworkers know that I enjoy automating myself out of my job, and sometimes in the past they have been surprised to learn that I haven't automated a task that by all measures is a prime candidate for automation. My answer is usually "Oh I plan to, I'm just not ready yet." The fact is that even if you have a task that is a great candidate for automation, it may not necessarily be the right time to automate it.

When I need to perform a new task that's a series of mundane, manual steps, I like to force myself to perform it step by step at least a few times "in the wild" before I start automating it. I find I usually need to perform a task a few times to understand where automation makes the most sense, what areas of the task may require extra attention, and what sorts of variables I might encounter for the task. Otherwise, if I just charge ahead and write a script, I may find myself rewriting it from scratch a few weeks later because I discover the process needs to be adapted to a new variation of the task. If I'm not quite sure about parts of a process, I may automate only the parts I am sure of first and get those right. Later on

when the rest of the process starts to gel in my mind, I then go back and incorporate it into the automation I've already completed.

I also avoid automating tasks if I'm not sure I can do so securely. For instance, a number of organizations are big fans of using ChatOps (automating tasks using bots inside a chatroom) for automation. Although I know that many bots can authenticate tasks before they perform them, I still worry about the potential for abuse with a service that's usually shared across the whole company, not to mention the fact that production changes are being triggered by a host outside the production environment. With my current threat model, I have to maintain strict separation between development and production environments, so having a bot accessible to anyone in the company, or having a Jenkins continuous integration server in the development environment performing my production tasks, just doesn't work. In many cases, I have fully automated tasks up to the point that it still requires an administrator with the proper access to go to the production environment (thereby proving that they are authorized to be there) before they push "the button".

How You Should Automate

Since the whole goal of automation is to save time, I don't like to waste time refactoring my automation. If I don't feel like I understand a process and its variables well enough to automate it, I wait until I do or automate only the parts I feel good about. In general, I'm a big fan of building a foundation of finished work that I then build upon. I like to start with automating tasks that will give me the biggest time-savings or

encourage the most consistency and then build off them.

I like doing the hard work up front so that it's easier down the road, and that is why I am a big fan of configuration management to automate server configuration. Once something like that is in place, rolling out changes to configuration becomes trivial, and creating new servers that match existing ones should be easy. These big tasks may take time up front, but they provide huge cost savings from then on, so I try to automate first.

I also favor automation tasks that can be used in multiple ways down the road. For instance, I think all administrators these days should have a simple, automated way to query their environment for whether a package is installed and on what hosts, and then be able to update that package easily on the hosts that have it. Some administrators refer to this as part of orchestration. (See my articles "Orchestration with MCollective, Part I and Part II" for more information on orchestration.)

Package updates are something that sysadmins do constantly both for in-house software that changes frequently and system software that needs security updates. If a security update is a burden, many sysadmins won't bother. Having automation in place to make package updates easy means administrators save time on a task they have to perform frequently. Sysadmins then can use that automated package update process both for security patches, in-house software deployments and other tasks where package updates are just one component of many.

As you write your automation, be careful to check that your tasks succeeded, and if not, alert the sysadmin to the

problem. That means shell scripts should check for exit codes, and error logs should be forwarded somewhere that gets the administrator's attention. It's all too easy to automate something and forget about it, but then check back weeks later and discover it stopped working!

In general, approach automation as a way to free up your brain, time and expertise toward tasks that actually need them. For me, I find that means time spent improving automation and otherwise dealing with exceptions—things that fall outside the normal day. If you keep it up, you eventually will find that when there are no crises or new projects, the day-to-day work should be automated to the point that your task is just to keep an eye on your well-oiled machine to make sure everything's running. That is when you know you have replaced yourself with a shell script.

Ticketing

This is the third in a series of chapters on system administrator fundamentals where I focus on some lessons I've learned through the years that might be obvious to longtime sysadmins, but news to someone just coming into this position.

In the first chapter, I discussed how to approach alerting and on-call rotations as a sysadmin. The second chapter covered how to automate yourself out of a job. In this chapter, I explore something that on the surface may seem boring or mundane but is absolutely critical to get right if you want to be an effective sysadmin: ticketing.

By ticketing, I'm referring to systems that allow sysadmins to keep track of tasks both internally and those requested by their coworkers or customers. There are many ways to get ticketing wrong so that it becomes a drain on an organization, so many sysadmins avoid or it use it begrudgingly. Also, ticketing approaches that work well for developers may be horrible for sysadmins, and vice versa. If you don't currently use a ticketing system, I hope by the end of this chapter, I've changed your mind. If you do use tickets, but you wish you didn't, I hope I can share how to structure a ticketing system that makes everything easier, not more difficult.

Why Tickets Are Important

Like documentation, tickets are one of those important things

in a mature organization that some administrators think are unnecessary or even a waste of time. A ticketing system is important no matter the size of your organization. In a large organization, you have a large volume of tasks you need to keep track of distributed among a group of people. In a small organization, you often have one person taking on many roles. This leads me to the first reason why tickets are important.

Tickets Ensure That Tasks Aren't Forgotten

Sysadmins are asked to do new tasks constantly. These days, there are any number of ways a coworker might ask for your help, from an email, to a phone call, to a message in a chat program, to a tap on the shoulder. If you weren't doing anything else, you immediately could start working on that task, and everything would be fine. Of course, usually sysadmins have to balance needs from many different people at the same time. Even requests that come in through email have a tendency to fall through the cracks and be forgotten. By storing every request in a ticket, no matter how you got the request, it is captured, so that even if you do forget about it, you'll remember it the next time you look at your ticketing system.

Tickets Make Sure the Task Is Done Right

Even if you can remember what someone wants you to do, you may not remember on Monday all of the details that someone told you in person on Friday. A ticket lets you capture exactly what people want done in their own words and provides a way

for them to confirm that you completed the task the way they wanted before you close the ticket.

Tickets Help You Prioritize Tasks

Every request is important to the person who makes it. Every request may not be as urgent to you or your team compared to your other tasks, however. When all of your tasks are captured in tickets, the team lead or manager can go through and re-prioritize tasks so they are worked on in the right order. This ends up being more fair for everyone; otherwise, new tasks have a way of cutting in line, especially when the person asking for something is standing over your shoulder.

With a ticketing system, team leads or managers have a full list of important tasks they can point to when they need to explain why they aren't dropping everything for a new request. At the very least, it will help direct the conversation about why a particular task should be put at the head of the line.

Tickets Distribute the Work

If you have only one sysadmin, distributing tickets and projects is easy. Once your team grows though, it's important to distribute the work so no member of the team gets burned out. Coworkers have a tendency of finding that senior member of your team who is most productive and going to them directly when they have any issue. Of course, that team member is probably already working on plenty of other tasks or may be trying to focus on an important project.

When a task is captured in a ticket, the team lead or

manager can assign and reassign tickets to different members of the team to make sure no one gets burned out, and also to ensure that everyone learns how to do things. Otherwise, you end up cultivating specialists within the team that always take tickets related to certain systems, which leads to problems later when that team member goes on vacation.

Tickets Provide an Audit Trail for Changes

Every time you change a system, you create an opportunity for something to break. If you are lucky, things break immediately after you make the change. More often, you'll find that it takes some time for a change to cause a problem. You'll discover two weeks later that something stopped working, and with a ticketing system, you can pull up all of the tasks that were worked on around that time. This makes it much easier to pinpoint potential causes of a problem.

Tickets also provide an audit trail for tasks that require approval or proof of completion, like creating or revoking accounts, granting new privileges or patching software. When someone asks who said it was okay for Bob to get access to production and when it happened, you can answer the question. If you need to prove that you applied a security patch, you can point to command output that you capture and then store in the corresponding ticket.

Qualities of an Effective Ticketing System

Many different ticketing systems exist, and sometimes when you hear people complain about tickets, what they are really complaining about is a bad ticketing system. When choosing

between ticketing systems, you should look for a few things.

Some systems that developers use to track code through the development process result in very complicated workflows. For a sysadmin though, the simpler the ticketing system the better. Because you already are asking a sysadmin to take time out of solving a problem to document it in a ticket, it helps if the ticketing process is fast and simple. I prefer very simple ticket workflows for sysadmins where there may be only a few states: open, assigned, in progress, resolved and closed. (I'll talk more about how I treat each of those states in the next section.)

The fewer required fields in a ticket, the better. If you want to add extra fields for tags or other information, that's fine, just don't make those fields mandatory. The goal here is to allow sysadmins to create tickets based on someone walking up and tapping them on the shoulder in less than a minute.

Ideally, the ticketing system would allow you some other way to generate tickets from a script, either from sending an email to a special address or via an exposed API. If it has an API that lets you change ticket state or add comments, all the better, as you potentially can integrate those into your other automation scripts. For instance, I've created a production deployment script that integrates with my ticketing system, so that it reads the manifest of packages it should install from the ticket itself and then outputs all of the results from the deployment as comments in the ticket. It's a great way to enforce a best practice of documenting each of your software releases, but it does it in a way that makes it the path of least resistance.

Favor ticketing systems that allow you to create

dependencies or other links between tickets. It's useful to know that task A depends on task B, and so you must complete task B first. These kinds of ticketing systems also make it easier to build a master ticket to track a project and then break that large project down into individual tickets that describe manageable tasks. These kinds of systems often show all of the subordinate tickets in the master ticket, so a quick glance at the master ticket can give you a clue about where you are in a project.

How to Manage and Organize Tickets

Each ticketing system has its own notion of ticket states, but in my opinion, you should, at a minimum, have the following:

- Open: a task that needs to be completed, but hasn't been assigned to anyone.
- Assigned: a task that's in a particular person's queue, but they haven't started work on it yet. Tasks in this state should be safe to reassign to someone else.

- In progress: a task that has been assigned to someone who is currently working on the task. You definitely should communicate with the assignee before you reassign tickets in this state.

- Resolved: the sysadmin believes the task has been completed and is waiting for confirmation from the person who filed the ticket before closing it.

- Closed: the task has been completed to everyone's satisfaction.

A well run ticketing system should provide the team with the answers to a few important questions. The first question is "What should I work on now?" To answer that question, each member of the team should be able to claim tickets, and team leads or managers should be able to assign tickets to individual members of the team. It's important for people to claim tickets and start work only after they are claimed; otherwise, it's easy (and common) for two members of the team to start working on the same task without realizing it. Then everyone on the team can start working on tickets in their personal queue, starting with the highest-priority tasks.

The next question a good ticketing system should answer is "What should I work on next?" Once sysadmins' personal queues are empty, they should be able to go to the collective queue and see a list of tasks ordered by priority. It should be clear what tasks they should put on their queue, and if there's any question about it, they can go to the team lead or manager

for some clarity. Again, ticket priority helps inform everyone on the team about what's next—higher-priority tasks trump lower-priority ones, not necessarily because they are less important (a ticket is always important to the person who filed it), but because they are less urgent.

I approach ticket priority as a way for users to help inform the team about how important the ticket is to them, but not how urgent it is for the team. The fact is, there's no way every employee in the company can know all of the other important tasks the sysadmin has to perform for other people nor can they be expected to weigh the importance of their need against everyone else's needs.

A good manager should reserve the right to weigh the priority assigned to a ticket against the other tickets in the queue and change the priority up or down based on its urgency relative to the other tasks. It also may be the case where a task that was low urgency two weeks ago has become urgent now because of how long it was in the queue, so a good manager would be aware of this and bump the priority. If you are going to start the practice of changing ticket priorities though, be sure to inform everyone of your intentions and how you will determine the urgency of a ticket.

Another key to managing tickets is to make sure all of your requests are captured in the ticketing system. Sometimes a coworker can be guilty of trying to skip ahead in line by messaging you with a request or walking directly to your desk to ask you to do something. Even in those cases where you really are going to drop everything to work on their request, you should insist on capturing the request in a ticket so you can track the work. This isn't just so you can prioritize it based

on other tasks or so you don't forget it, it's so in a week when some problem crops up based on this urgent change, you'll see this ticket along with other tasks completed that day and it will help you track down the cause.

Finally, as a manager, be careful to distribute work fairly among your team. Even if one member of the team happens to be an expert on a particular service, don't assign that person every task related to that service; it's important for everyone on the team to cross-train. Pay attention if employees try to get tickets assigned to their favorite member of the team, and don't be afraid to reassign tasks to spread the work around evenly. Finally, every ticket queue has routine, mundane grunt work that must be done. Be sure to distribute those tasks throughout the team so no one gets burnt out.

Patch Management

In this chapter, I cover some of the fundamentals of patch management under Linux, including what a good patch management system looks like, the tools you will want to put in place and how the overall patching process should work.

What Is Patch Management?

When I say patch management, I'm referring to the systems you have in place to update software already on a server. I'm not just talking about keeping up with the latest-and-greatest bleeding-edge version of a piece of software. Even more conservative distributions like Debian that stick with a particular version of software for its "stable" release still release frequent updates that patch bugs or security holes.

Of course, if your organization decided to roll its own version of a particular piece of software, either because developers demanded the latest and greatest, you needed to fork the software to apply a custom change, or you just like giving yourself extra work, you now have a problem. Ideally you have put in a system that automatically packages up the custom version of the software for you in the same continuous

integration system you use to build and package any other software, but many sysadmins still rely on the outdated method of packaging the software on their local machine based on (hopefully up to date) documentation on their wiki. In either case, you will need to confirm that your particular version has the security flaw, and if so, make sure that the new patch applies cleanly to your custom version.

What Good Patch Management Looks Like

Patch management starts with knowing that there is a software update to begin with. First, for your core software, you should be subscribed to your Linux distribution's security mailing list, so you're notified immediately when there are security patches. If there you use any software that doesn't come from your distribution, you must find out how to be kept up to date on security patches for that software as well. When new security notifications come in, you should review the details so you understand how severe the security flaw is, whether you are affected and gauge a sense of how urgent the patch is.

Some organizations have a purely manual patch management system. With such a system, when a security patch comes along, the sysadmin figures out which servers are running the software, generally by relying on memory and by logging in to servers and checking. Then the sysadmin uses the server's built-in package management tool to update the software with the latest from the distribution. Then the sysadmin moves on to the next server, and the next, until all of the servers are patched.

There are many problems with manual patch management. First is the fact that it makes patching a laborious

chore. The more work patching is, the more likely a sysadmin will put it off or skip doing it entirely. The second problem is that manual patch management relies too much on the sysadmin's ability to remember and recall all of the servers he or she is responsible for and keep track of which are patched and which aren't. This makes it easy for servers to be forgotten and sit unpatched.

The faster and easier patch management is, the more likely you are to do it. You should have a system in place that quickly can tell you which servers are running a particular piece of software at which version. Ideally, that system also can push out updates. Personally, I prefer orchestration tools like MCollective for this task, but Red Hat provides Satellite, and Canonical provides Landscape as central tools that let you view software versions across your fleet of servers and apply patches all from a central place.

Patching should be fault-tolerant as well. You should be able to patch a service and restart it without any overall down time. The same idea goes for kernel patches that require a reboot. My approach is to divide my servers into different high availability groups so that lb1, app1, rabbitmq1 and db1 would all be in one group, and lb2, app2, rabbitmq2 and db2 are in another. Then, I know I can patch one group at a time without it causing downtime anywhere else.

So, how fast is fast? Your system should be able to roll out a patch to a minor piece of software that doesn't have an accompanying service (such as bash in the case of the ShellShock vulnerability) within a few minutes to an hour at most. For something like OpenSSL that requires you to restart services, the careful process of patching and restarting

services in a fault-tolerant way probably will take more time, but this is where orchestration tools come in handy. I gave examples of how to use MCollective to accomplish this in my MCollective orchestration articles (see Part I and Part II), but ideally, you should put a system in place that makes it easy to patch and restart services in a fault-tolerant and automated way.

When patching requires a reboot, such as in the case of kernel patches, it might take a bit more time, but again, automation and orchestration tools can make this go much faster than you might imagine. I can patch and reboot the servers in an environment in a fault-tolerant way within an hour or two, and it would be much faster than that if I didn't need to wait for clusters to sync back up in between reboots.

Unfortunately, many sysadmins still hold on to the outdated notion that uptime is a badge of pride—given that serious kernel patches tend to come out at least once a year if not more often, to me, it's proof you don't take security

seriously.

Many organizations also still have that single point of failure server that can never go down, and as a result, it never gets patched or rebooted. If you want to be secure, you need to remove these outdated liabilities and create systems that at least can be rebooted during a late-night maintenance window.

Ultimately, fast and easy patch management is a sign of a mature and professional sysadmin team. Updating software is something all sysadmins have to do as part of their jobs, and investing time into systems that make that process easy and fast pays dividends far beyond security. For one, it helps identify bad architecture decisions that cause single points of failure. For another, it helps identify stagnant, out-of-date legacy systems in an environment and provides you with an incentive to replace them. Finally, when patching is managed well, it frees up sysadmins' time and turns their attention to the things that truly require their expertise.

Leveling Up

In this chapter, I describe the overall sysadmin career path and what I consider the attributes that might make you a "senior sysadmin" instead of a "sysadmin" or "junior sysadmin", along with some tips on how to level up.

Keep in mind that titles are pretty fluid and loose things, and that they mean different things to different people. Also, it will take different people different amounts of time to "level up" depending on their innate sysadmin skills, their work ethic and the opportunities they get to gain more experience. That said, be suspicious of anyone who leveled up to a senior level in any field in only a year or two—it takes time in a career to make the kinds of mistakes and learn the kinds of lessons you need to learn before you can move up to the next level.

Junior Systems Administrator

Junior sysadmins are early on in their sysadmin training. It might be their first sysadmin job where they are learning everything from scratch, or they might have a few years of experience under their belts. Either way, a few attributes are common among junior sysadmins:

- Tasks will require help from other members of the team to complete.

- They will rely heavily on documentation and may not understand what individual tasks do.
- It may take weeks or even months to be productive at a new job.
- Most of their time will be spent with daily tickets.
- Eventually they might take on a project, but will need quite a bit of help to complete it.

One of the first attributes that defines junior sysadmins is the amount of outside help they will need to do their jobs. Generally speaking, they will need help and direction to perform day-to-day tasks, especially at first. If you document your routine tasks (and you should!), you will find that junior sysadmins will dutifully follow your procedures step by step, but they may not understand exactly what those steps do. If a task deviates from the norm, or if for some reason a step fails, they will escalate up to a more senior member of the team for help—this is a good thing, because this mentoring is one of the main ways that junior sysadmins build their experience besides making mistakes and fixing them.

It might take sysadmins at this level a few weeks or even months at a new organization until they are productive and can start doing daily tasks independently without help. These are great opportunities for a team to audit documentation and for junior members of the team to flag gaps in documentation or places where they are out of date. If you have junior team

members add documentation themselves, just make sure that a more senior team member goes over it to make sure it's correct and complete.

A sysadmin's task list is usually divided into two main categories: day-to-day tasks and projects. Junior sysadmins often end up being assigned more of the day-to-day "grunt work", not as a punishment, but just because projects usually require more experience—experience they will get as they master daily tickets.

That said, at some point, it will be important for junior sysadmins to take on their first project. Ideally, this will be a project without a strict deadline, so they can take the time they need to research and get it right. At this level, a more senior team member will need to devote a fair amount of time to act as a mentor and help direct the planning and research for the project and answer any questions.

Both daily tasks and projects are important for junior sysadmins, as it's the mastery of daily tasks and the successful completion of a couple projects that will help prepare junior sysadmins to level up. Each task they master will add a certain level of confidence and proficiency in routine sysadmin tasks, and projects will help develop their research skills and the ability to complete tasks that fall outside a playbook.

Mid-Level Systems Administrator

It can be difficult to draw the exact line where a sysadmin levels up past the junior level. There isn't an exact number of years' experience needed; instead, it has more to do with sysadmins' competency with their craft and their overall confidence and independence.

Here are a few attributes that are common to mid-level sysadmins:

- They generally perform day-to-day tasks independently.
- They understand some of the technology behind their routine tasks and don't just parrot commands they see in documentation.
- It takes a few weeks up to a month to be productive at a new job.
- Their time is pretty equally balanced between daily tickets and longer-term projects.
- They are able to come up with new approaches and improvements to existing tasks.
- They can complete simple projects independently and

more complex projects with some help from more senior team members.

The main difference between junior sysadmins and mid-level sysadmins has to do with their independence. As sysadmins become more comfortable with servers in general, and the processes within an organization specifically, they start to be able to perform typical tasks by themselves. Mid-level sysadmins should be able to handle all of the normal tasks that are thrown at them without outside help. It's only when they get an odd "curve ball", such as a one-off task that hasn't been done before or some unique emergency, that mid-level sysadmins may need to reach out to the more senior members of the team for some guidance. As with junior sysadmins, this type of help is very important, and it would be a mistake for mid-level sysadmins not to ask for help with odd requests just to try to be "more senior". Asking questions and getting advice from more experienced sysadmins will help them level up. If they try to go it completely alone, no matter what, it will take much longer.

Mid-level sysadmins also take on more projects than their junior counterparts, and they are able to complete simple projects independently. Junior sysadmins might be able to maintain an existing system, but mid-level sysadmins actually might be able to set it up from scratch. They also can start tackling larger, more complicated projects that may require them to learn new technologies and come up with some approaches independently, although in those cases, they'll still sometimes need to reach out to more experienced team members to make sure they are on the right track.

As sysadmins master all of the day-to-day tasks, they also naturally will start to come up with improvements and efficiencies for those tasks, and they may make some suggestions to the team along those lines. These improvements may become projects for them in their own right. They also should be able to provide some level of mentorship and training for junior members on the team, at least with daily tasks.

One of the most important things for mid-level sysadmins to do if they want to level up is to take on projects and help triage emergencies. Projects and emergencies often provide opportunities to think outside established playbooks. It's this kind of critical thinking, research and problem-solving that builds the experience that's so important for sysadmins. They will start to notice some common patterns the more emergencies and projects they work through, and that realization builds a certain level of confidence and deeper understanding that is vital for moving to the next level.

Senior Systems Administrator

Although some may consider people to be senior sysadmins based on a certain number of years' experience, to me, what makes someone a senior sysadmin versus a mid-level sysadmin isn't years of experience or number of places worked at, it's more a particular state of mind that one can get to via many different means. Many people get the title before they get the state of mind, and often it takes getting the title (or some of the responsibilities associated with it) to make a person level up.

The main difference between senior sysadmins and mid-level sysadmins is that one day, something clicks in senior sysadmins' minds when they realize that basically every emergency they've responded to and every project they've worked on to that point all have a common trait: given enough time and effort, they can track down the cause of just about any problem and complete just about any sysadmin task. This is a matter of true confidence, not false bravado, and it's this kind of real independence that marks senior sysadmins.

Early on in your career, certain tasks or projects just seem over your head, and you absolutely need help to complete them. Later on, you master daily tasks, but weird emergencies or complex projects still may intimidate you. Senior sysadmins have completed so many projects and responded to so many emergencies, that they eventually build the confidence such that they aren't intimidated by the next project, the next emergency or the prospect of being responsible for important mission-critical infrastructure. Like mid-level sysadmins might approach their daily tickets, senior sysadmins approach any task that comes their way.

Here are some attributes common to senior sysadmins:

- They can perform both daily tasks and complex projects independently.

- They understand the fundamentals behind the technologies they use and can distill complex tasks down into simple playbooks everyone on the team can follow.

- They can be productive at a new job within a week or two.
- Their time is spent more on large projects and odd requests that fall outside the norm.
- They mentor other team members and have a good sense of best practices.
- They come up with new projects and improvements and can suggest appropriate designs to solve new problems.
- They understand their own fallibility and develop procedures to protect themselves from their own mistakes.

Again, it's the confidence and independence of senior sysadmins that separates them from mid-level sysadmins. That's not to say that senior sysadmins never ask for help. Along with the confidence of being able to tackle any sysadmin task is the humility that comes with a career full of mistakes. In fact, part of their experience will have taught them the wisdom of asking other people on the team for feedback to make sure they haven't missed anything. Often senior sysadmins will come up with multiple ways to tackle a problem, each with pros and cons, and use the rest of the team as a sounding board to help choose which approach would work best in a specific case.

Senior sysadmins' experiences exposes them to many different technologies, systems and architectures through the years. This means they start to notice which approaches work,

which don't, and which work at first but cause problems in the long run. In particular, they might track some project they completed themselves through its lifetime and note how their initial solutions worked to a particular point and then either failed as it scaled, or needed to change with the advent of some new technology. The more this happens, the more senior sysadmins start to develop a natural sense of best practices and what I call the "sysadmin sense", which, like Spiderman's "spidey sense", starts to warn them when they see something that is going to result in a problem down the road, like a backup system that's never been tested or a system that has a single point of failure. It's in developing this expertise that they are able to level up to the last major level outside management.

Systems Architect

Although every organization is a bit different, there are two main career paths senior sysadmins might choose from as they gain experience. The most common path is in management. Senior sysadmins over time end up spending more time mentoring their team and often are promoted to team leads and from there into full managers over their teams. The other path continues on with the "individual-contributor" role where they may or may not act as team leads, but they don't have any direct reports and don't spend time doing employee evaluations or things of that sort. Of course, there also are paths that blend those two extremes. In this last section, I describe one of the last levels for an individual-contributor sysadmin to move to: systems architect.

In many organizations, the line between a systems architect and a senior sysadmin can be blurry. Equally blurry

are the qualifications that may make someone a systems architect. That said, generally speaking, systems architects have spent a number of years as senior sysadmins. During the course of their careers, they have participated in a large number of projects, both with a team and independently, and they have started to see what works and what doesn't. It's this accumulation of experience with a wide variety of technologies and project designs that starts to build this inherent sense of best practices that makes someone a systems architect.

The following are some attributes common to systems architects:

- They are familiar with many different technologies that solve a particular problem along with their pros and cons.
- When solving a problem, they come up with multiple approaches and can explain and defend their preferred approach.
- They understand the limitations to a solution and where it will fail as it scales.
- They can distill a general problem down to individual tasks as part of a larger project that can be divided among a team.
- They can evaluate new technologies based on their relative merits and not be distracted by hype or

popularity.

Systems architects aren't necessarily married to a particular approach, although they may have a set of approaches for tackling certain problems based on what's worked for them in the past. Because they have operated at a senior level for some time, they have developed a deeper understanding of what defines a good architecture versus a bad one and how to choose one technology over the other. Technology moves in trends, and those trends tend to repeat themselves over a long enough timeline. Systems architects have been around long enough that they have seen at least one of those trend cycles for some hyped technology, and they probably have been burned at some point in the past by adopting an immature technology too quickly just because it was popular. Whereas junior administrators are more likely to get caught up in the hype behind a particular new technology and want to use it everywhere, systems architects are more likely to cut through the hype and, for any new technology, be able to identify where it would be useful and where it wouldn't.

Conclusion

I hope this description of levels in systems administration has been helpful as you plan your own career. When it comes to gaining experience, nothing quite beats making your own mistakes and having to recover from them yourself. At the same time, it sure is a lot easier to invite battle-hardened senior sysadmins to beers and learn from their war stories. I hope this ebook on sysadmin fundamentals has been helpful for those of you new to the sysadmin trenches, and also I hope it helps save you from having to learn from your own mistakes as you move forward in your career.