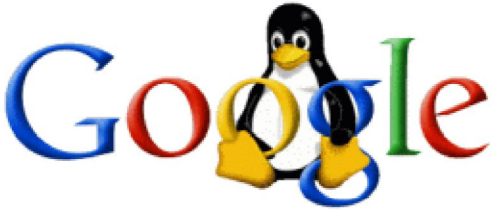
Linux.conf.au 2020



Planning for and avoiding failures

Slides: http://marc.merlins.org/linux/talks/Failure_Planning/

Questions: please wait until the end (many slides to go through)
Slides contain more content that the talk will have time to go into.

Marc MERLIN
merlin@google.com

Those who don’t learn from history are doomed to repeat it

- How often are we repeating the same programming mistakes?
- How often are sysadmins taking the mistaken shortcuts?
- Do you wish for “I should have known better” before the problem happened?
- What can we learn from the medical and aviation community?
- Never again will you walk by any “it this right?”, or “looks like I fixed the symptom, hopefully the problem won’t happen again”
- This talk will look at various failures across fields to encourage you to learn from them and have less of a need to create new ones :)
- Hopefully you can grow a spidey sense that will help you in the future.

If doesn’t feel quite right, it probably isn’t

- How often have you dealt with people who said “ok, the problem went away, so it’s solved, I can go back to whatever else I was doing”


NOOOOO!!!!
ICANHASCHEEZBURGER.COM

If it’s not root caused, it’s not fixed

- Yes, you have other things that need you, but they should wait..
- … until you fully understand what went wrong and why
- If you can’t fix it right now, have a bug for you or someone else to fix soon
- Fixing symptoms is never an appropriate fix unless you understand the underlying problem and it can’t be fixed right away
- “It went away, so we’re all good” is never a proper answer
- I deal with this all the time with support people. Please set a higher bar for yourself and others by example
- Don’t reward or measure people by amounts of cases closed or bugs fixed.

There are only 2 kinds of mistakes you can make

1) Honest ones that would have been hard to plan for

2) You being too proud / impatient / ”I know what I’m doing” / “This should (probably) work fine”

3) All the other kinds :)

#1 Is non trivial to fix but #2 can really be fixed with a single slide

The one slide that will prevent many failures

- Have you ever watched the old US TV show ER?
- When a doctor or intern lost a patient they were “invited” to a post mortem (a real one in this case)
- They were grilled on what they had done and hadn’t done and why
- Each time you are making a programming or sysadmin decision that you question just a little bit, ask yourself:

  “if this is all blows up, will my rationale for this decision make sense? Will people agree I was careful enough and didn’t miss an obvious better option?
- Will they (or I) not judge me for having acted impulsively?

When does this apply?

- Every time you question whether it should :)
- Each time you are taking a shortcut
- Doesn’t mean you should never/can’t
- Just that you weighed the pros and cons carefully
- So, if it all blows up, at least it was a careful decision.
- Companies are supposed to have processes to keep you honest, but you are the best person positioned to do it.

Beware of the highs of quick progress and changes

- You’re getting things done, makes you feel good for achievement, self esteem, and possible high from this
- It’s great to make quick progress, not letting red tape get in the way
- But it pushes you to work alone, make more changes, cut corners, take more risks
- This is how people operate in many startups, you can get 2 or 3 times more things done.

Beware of the highs of quick progress and changes (2)

- Resulting risks can be acceptable to startups who do not have the resources to higher the amount of staff needed and rely on heroics and luck
- It’s a fun and addictive environment for some
- Lack of direction and reviews also cause people to spend effort on things that don’t matter in the end and make avoidable mistakes
- Those are not things any big company wants or can justify, they’d rather hire people who made those mistakes elsewhere and hopefully learned from them :)
- Watch “general magic” (2018) https://www.imdb.com/title/tt6849786/

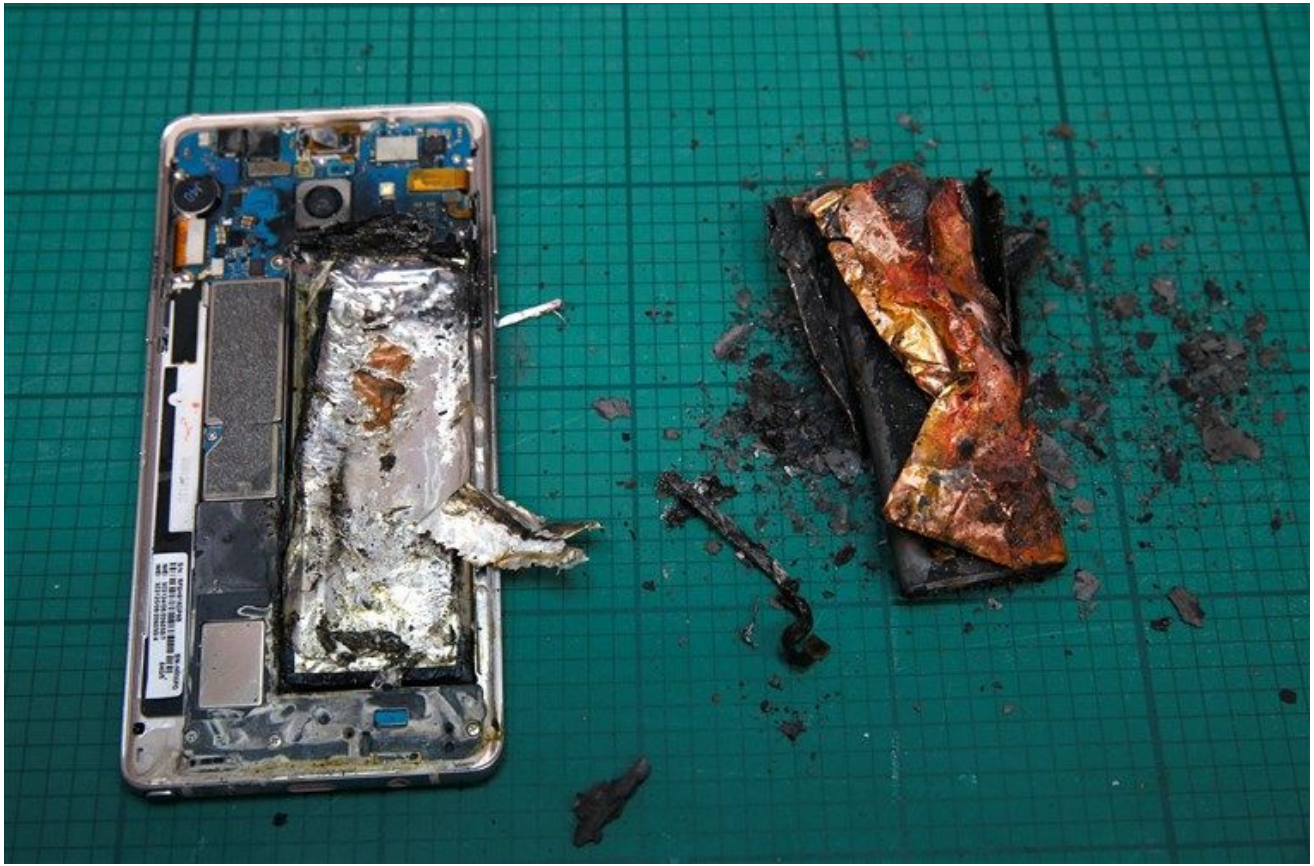
You’re done, thanks for coming

Ok, there is a little more

- But seriously, if you only remember the last 2 slides, you'll be so far ahead of most people.
- The rest of the talk will look at lessons in learned in multiple areas, so that you can save the trouble of learning them from scratch

Remember those Samsung Note 7s that self ignited?

- The battery was upgraded last minute before shipping
- It mostly fit, but there was no room for expansion
- Manufacturing used 2 different battery manufacturers, one had batteries that were ever so slightly more thick
- Some batteries expanded into a screw or other scrap metal left by other shoddy manufacturing
- which breached the protective envelope
- which in turn allowed the lithium to react with the air
- and happily burn
- NOT PROFIT!



Morals of the Samsung Note 7s Story

- Let’s be honest, this is a complex multi level failure
- Each component passed its own tests
- They likely even passed integration tests

Lessons learned:

- Design redundant/multi level safeties
- Like failsafes in the battery (thermal expansion, temperature, etc)
- If time/budget allows, don’t rely on end user testing only, but do include it as the last level
- Whenever possible, have flexible ship dates. **Don’t force ship something everyone knows isn’t sufficiently tested/ready/finished**, unless you really must (Xmas)

Software

Google Culture

- Revert first, discuss/point fingers later
- Code reviews or Change requests for ops
- Unittests
- Blameless post mortems
- Practice emergency/recovery

Code Reviews

- No matter how good we are, we all make mistakes eventually
- Just sending a code review, forces you to re-read your changes and think about a justification for all of them
- The amount of bugs found from sending code reviews, is staggering
- If you work solo, do a self code review when writing git checkin message
- This is just like the amount of problems I fixed on my own while writing a detailed Email on what was wrong and what I had already tried
- The first reviewer of your code sent for review should always be you!
- Pair programming can used in lieu of TBR

Change Requests

- For (Sys)Ops, sending a plan of what you're going to change and why works similarly.
- Including why it's best to do it in the middle of the night/weekend (downtime for users)
- Weighed against why it's not (staff tired and motivated to go back to bed or weekend ASAP, not around to deal with delayed problems)
- Impact on other teams that may not be around/awake to notice breakage if not done during work hours (with suitable warning of course)

TBR: To Be Reviewed

- When time matters and “We really need to submit this, and I’m putting my job on the line for it”, we have TBRs.
- Your code still gets reviewed, but you can submit it before it is
- Obviously you should be ready to justify why you circumvented the normal process
- If you do this too often, your manager will likely have a chat with you, or even if you used it a single time but at a very wrong time.
- With great power comes great responsibility.
- This works surprisingly well. Checks and Balances...

Unittests

- By now, most serious companies and big projects use unittests.
- They are necessary but don’t treat them as sufficient
- Sometimes unittests are nothing more than compile tests, try to strive for more
- Aim for end to end functional tests, using an emulated environment that mirrors the real one as much as possible (I sent ‘2’ in the function, and it sent back ‘3’ is a very low bar).
- Try to use a real system instead of mocking it away, if possible
- They are great for testing on architectures the developer doesn’t own

CQ/CI (Commit Queue, Continuous Integration)

- Catch regressions early to make them easier to find. I've seen too many problems that took so long to find after they were allowed in the tree due to lack of tests. Have tests and CI/CQ as soon as you can.
- Projects like Chrome/ChromeOS and Fuchsia use both, based on luci https://luci-scheduler.appspot.com/jobs/fuchsia/ci-fuchsia-x64-release
- Developers are supposed to do their own testing first and before they can send a change for review, it is run by CQ, which for fuchsia runs lots of tests in emulated environments (qemu in google cloud), and actual hardware of different types (CPU and peripherals)

CQ/CI costs and issues

- Hardware tests are often more expensive to scale, so tests are targeted accordingly
- Some tests are too expensive/too long to run as presubmit/CQ, so they are run in CI
- If a CI run fails, it notifies the list of CIs that were included, the less often CI can be run, the more people get bundled in that warning Email, and probably confused.
- Running tests on real hardware is essential for projects like ChromeOS or Fuchsia, and useful for some projects like Chrome that use hardware to find some bugs, timing or power use regressions, but requires a dedicated team to roll out and maintain the hardware
- You now also need to deal with both software and hardware flakes. Retrying tests once helps, after that you need people to look into why they flake or failed.
- Aggressively go after flaky software tests and get them fixed or removed. A flaky test is worse than no test.

Percent rollouts / Safe file updates

- Anything that rolls out a text file (like /etc/aliases), should refuse to proceed if more than x% is being changed (catches accidental partial deletes)
- Just like unittests, make sure new config files have proper syntax and the intended behaviour before they get rolled out
- Back in the day, I wrote a DNS/bind config edit system that spun a test bind with the new config files, scanned logs for warnings and errors, and did test queries against it before the new config file was allowed to be checked in and rolled out (functional unittest with real parser instead of mocked one)

Blameless Post Mortems

- https://landing.google.com/sre/sre-book/chapters/postmortem-culture/
- When things go wrong anyway (not if, but when), revert first, argue later
- Anyone can request a post mortem
- Either because there is truly something to learn
- Sometimes also because the team(s) that made the mistake could use the experience and the time to reflect :)
- They state facts and timelines, not fault or blame.
- It’s often a bit humbling to have to write a postmortem, so, it’s also an incentive to try harder to avoid a future one :)
- Failure should be treated as a lesson for all to learn from
- “Why should I fire you? I just spent $10M educating you” (Watson, IBM)

https://andrewsrandomthoughts.wordpress.com/2012/07/29/the-10-million-dollar-mistake/

Practice Emergencies when everyone is awake and ready

- We have a week per year when internal systems get to practice live failure and recovery scenarios (within reason)
- This includes having the main office swallowed by a major earthquake, and recovery having to be done by remote offices only
- These exercises often find documentation shortfalls and dependencies/compound failure scenarios that we didn't know about.
- They need to be done carefully though, you don't want to cause more damage than the potential for learning. This includes an abort/revert plan.
- That's an occasion to define and practice your revert scenario (any rollout needs to have a revert plan. Run forward only should avoided).

Chmod-Gate

- mkdir -p -m 755 /usr/local/foo is safe, is it not?
- It started taking down part of google production before someone pushed the emergency switch and stopped the rollout
- TL;DR: /usr/local/foo is created with 755 but then the user umask is applied (often 0027 or 0077 for safety).
- During testing, the tester had permissions to traverse /usr/local
- At install time, /usr/local/ got created with permissions 700 and broke everything as non root processes got access denied.
- Ultimately it’s all about relative speed of rollout, watching and being able to stop and revert quickly.

Overachieving engineers and computers

https://landing.google.com/sre/sre-book/chapters/automation-at-google/#xref_automation_diskerase-sidebar

- We have a few servers in some colos we don't own for various reasons
- To protect user and our data, we have a diskerase process.
- After trying to re-erase a rack that didn't seem fully erased, automation determined the new set of machines to erase was empty
- Due to a bug, that empty set was treated as "everything"
- The nice automation complied and did so quicker than humans were able to stop it.
- Moral of the story: double check any destructive code and make sure it gets throttled.

Percent rollouts are not the ultimate safety

- Google uses percent rollouts within datacenters and within the global network, they've always worked fine
- Until…
- One system rolled out a new config file that specified a rollout
- But it ended up having a problem that crashed every rollout server that received the file
- The percent rollout didn't save us and our efficiency in syncing within our network, cost us dearly
- Fix is to also gate rollout of config file changes

Learning From Post Mortems

- If you have to write it, makes it harder to wave off a mistake and do it again later
- If you get to read it, you will have those “I’m so glad I wasn’t the one who made that mistake” moments
- And there are those where a string of reasonable looking actions cause an unexpected failure. Those are the best ones to learn from and shape a more critical mind and your spidey sense
- “oh no, let’s not do this, team X did last year and turned out it was unsafe”
- https://landing.google.com/sre/sre-book/chapters/postmortem-culture/
- https://landing.google.com/sre/sre-book/chapters/reliable-product-launches/

My Personal Lessons

- From aviation and diving: have a plan before you need it, because once you're in the middle of problems, the bigger the problem, the more IQ you lose on average
- For flying/diving: learn and practice emergencies before they happen
- Be super careful about "temporary fixes" or "live fixes". There are times they are necessary, but no one involved should be allowed to go home before the fixes are submitted properly, or high priority bugs opened to fix it right. Also, **oncall people must be notified of the temporary situation**.

Summary

- Generally, a given person/company should make the same mistake twice, or you're really doing it wrong
- For extra points, you'll then also avoid doing similar mistakes you didn't have to make yourself but thought about in the post mortem, and/or read about
- For more extra points, you'll simply have read about that problem and avoided it without ever having to experience it yourself.
- For Ops/SRE, have a look at https://landing.google.com/sre/sre-book/ and other publications from others
- This is what aviation does a lot because relearning failures often leads to death
- Aviation says: "Experience is a cruel teacher, she gives the test first, and if you survive, then you get the lesson".
- We can learn from aviation!

A...

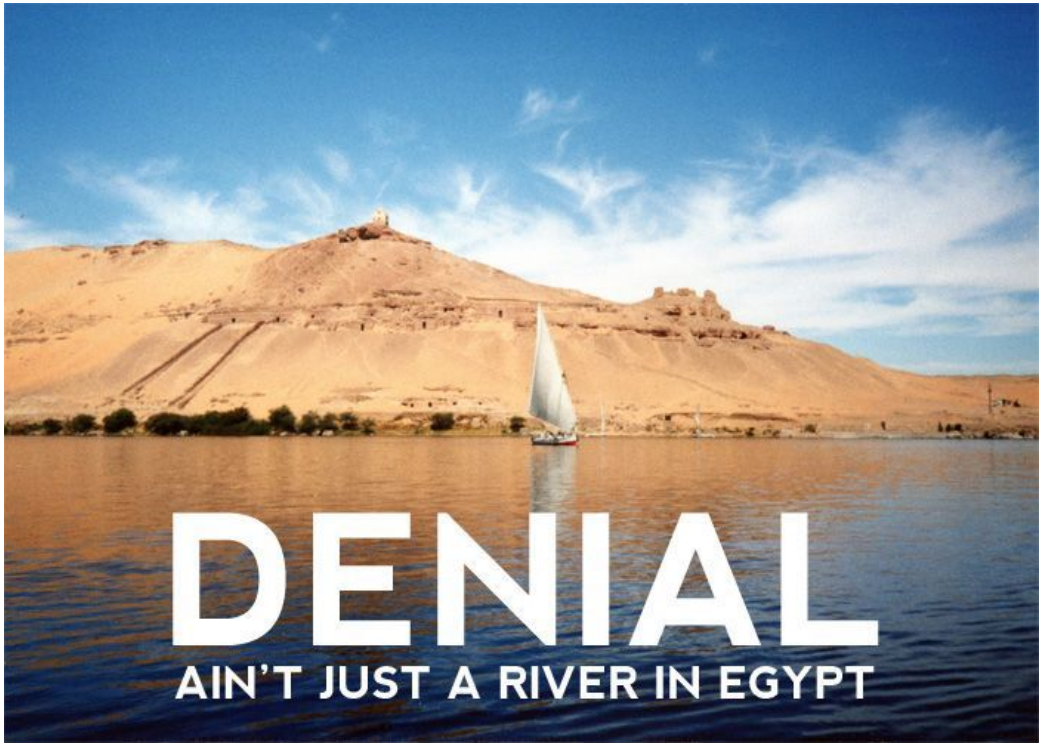
...via...

...tion

There is so much to learn from aviation

- I’m not a great pilot, I’m just an average one
- One thing I did learn though, was honest risk management
- Because it’s not about losing a bonus or getting a bad perf review
- It’s about not dying and not killing others either
- We are so good at rationalizing bad decisions while we are making them. It takes true willpower to break that
- Yet, our brains are really good at self denial

Remember, about self denial...



There is so much to learn from aviation (2)

- If the fear of a bad review or writing a postmortem doesn’t do it
- The fear of dying hopefully will
- And if not, then you remove yourself from the selection pool
- So it all works out :)


FIREFIGHTER: WHAT HAPPENED?
N36638
PILOT: I DON'T KNOW, JUST GOT HERE MYSELF.
imgflip.com


JOKES ABOUT CRASHES
ARE JUST PLANE WRONG
quickmeme.com

Airbus Automation and waning pilot skill: AF447

AF447, what went wrong

- 3 pilots, only one is seasoned, the other two are pretty junior
- Senior pilot goes for required sleep half way through the flight
- Bad thunderstorm over the equator, 3 pitot tubes iced up
- Plane automation turned off and reverted to alternate law
- Junior pilots poorly trained to actually fly a plane in alternate law and with no outside reference (no horizon)
- They can’t figure out how much to pull the nose up or down despite a standby instrument which admittedly is not easy to find in an emergency

AF447, what went wrong (2)

- Pilot pulls nose up so much that the plane enters a deep stall
- It gets so slow that the **stall warning turns off while in a very deep stall** because programmer thought a plane couldn't actually fly that slowly
- An airbus would never allow a pilot to do this, but in alternate law, it has to.
- Plane falls out of the sky, the least experienced pilot pulls the stick to continue the stall (rookie mistake) while the somewhat better one pushes it down
- Airbus with no warning to any pilot averages both inputs
- Plane stays in a deep stall and crashes in the ocean at 3000 meters/mn falling rate. Everyone dies, it takes over a year to find the black boxes.
- Airbus programmers said it was the pilots' fault for both trying to fly at the same time against procedures. Airbus planes still average input from 2 pilots with no warning to either pilot, today. **UI Matters!**

QF 32, Airbus A380 with terrible Rolls Royce Engines

- One of too many Roll Royce engine failures for that plane due to shoddy manufacturing.
- Uncontained engine failure that severs part of the wing, including some hydraulics and control to one engine
- Throws out around 100 errors on the plane computer.
- 4 pilots were kept busy acknowledging all those errors sent in random order and deciding which ones were important
- It took superior airmanship to barely land the plane. Everyone survived
- It’s unclear if airbus airplanes now prioritize errors in order or urgency

Check for bit flips, secret military bases, and solar flares

- Mayday Air Crash Investigation is an addictive series https://en.wikipedia.org/wiki/List_of_Mayday_episodes
- 2 planes had an unexplained uncontrolled pitch down towards the ocean in the same general area.
- Detailed and thorough investigation showed the angle of attack and speed data streams got crossed for a few samples and then crossed again.
- Computer received the wrong data samples and interpreted as stall imminent and did an emergency pitch down to recover.
- Planes self recovered because as the data streams recovered. Best guess was that a nearby base sent radio signals that created enough EMI to jam the planes' internal data buses.
- **Airbus programmers relied on their data bus being perfect and didn't plan for data corruption on the bus.**
- **It's easy to notice one data sample is out of wack and ignore it. Program defensively.**

737 Max, how many things went wrong?

- They forced everyone to pretend the plane was unchanged
- They didn't tell pilots about this critical system
- But yet, they expected pilots to diagnose the failure of that unknown system and know how to disable it in 4 seconds by pulling the correct fuse, or everyone would die
- 4 seconds is barely enough to diagnose and fix a rehearsed failure.
- The failure happened to untrained crews
- They didn't know what to do
- Everyone did die. Twice!
- All planes were finally grounded.

Boeing became unable to ship planes they were building

They had to halt production at 400 after running out of room

They had to pay billions to airlines with grounded planes

Agencies that do certifications vs complex systems

- The FAA washed their hands of all this complex computer stuff and allowed Boeing to “self certify”. Doesn’t it sound like “mmmh, I can’t understand your code, how about you do your own code review?”
- Will that remain the case? Is the FAA appropriately staffed?
- Have you seen Karen Sandler’s talks on her implanted defibrillator, how insecure they are and how easy it would be to remotely trigger them and kill their users?
- The FDA is no better than the FAA, they don’t sufficiently understand computer systems, and even still allow insecure windows XP medical devices connected to the internet!

When things unexpectedly go wrong

- Undo whatever was done last
- It doesn’t always work, but it has saved many airplanes
- It often works for computers too
- Never spend time arguing whose fault it is or why it should or shouldn’t be happening
- Revert first, talk later!
- If revert doesn’t help, try to fall back to pre-learned recoveries. Impressive recovery of a plane that was doomed to kill everyone https://youtu.be/WfNBmZy1Yuc?t=2031

Conclusions

- Learn from other people's experiences/mistakes.
- Continuous improvement, read tech magazines from your field (sysadmin, programming, pilot, etc...)
- If you see something, say something! Be a whistle blower!
- Think ahead for problems, grow your spidey sense
- Don't let people bully you out of your real concerns. Maybe you won't have a space shuttle blowup on your conscience for a problem you pointed out and was ignored, but don't let things go if you know they are wrong.
- Be honest with yourself, know if you are pushing past safety limits or taking too many risks. Be sure you can explain what you did or didn't with a straight face.
- For aviation, wouldn't it be nice if it were all open source so that researchers can help the FAAs of the world check for bugs? Did Airbus ever fix some of the software issues I mentioned? (never mind Boeing)

Q&A

Slides:
http://marc.merlins.org/linux/talks/Failure_Planning/

Marc Merlin
merlin@google.com