HACKERMONTHLY

Issue 8 January 2011

Curator
Lim Cheng Soon

Proofreaders
Lane Rapp
Jordan Greenaway
Millie Patel
D'vorah Elias

Printer
MagCloud

Contributors
ARTICLES
Tony Gebely
Brian Shul
Sahil Lavingia
Jason Cohen
Adrian Tan
J. Kenji Lopez-Alt
Derek Sivers
Hoyt Koepke
Thomas Roth
Gabriel Weinberg
Ben Strong
Greg Grothaus

COMMENTARIES
Jason Fried
Danilo Campos
Patrick McKenzie
Sushaantu
Michael F Booth
SDr
Tom Hughes-Croucher
Jon Pincus
Kliment Yanev

HACKER MONTHLY is the print magazine version of Hacker News — *news.ycombinator.com*, a social news website wildly popular among programmers and startup founders. The submission guidelines state that content can be "anything that gratifies one's intellectual curiosity." Every month, we select from the top voted articles on Hacker News and print them in magazine format. For more, visit *hackermonthly.com*.

Advertising
ads@hackermonthly.com

Contact
contact@hackermonthly.com

Published by
Netizens Media
46, Taylor Road,
11600 Penang,
Malaysia.

Cover Photo: Marcus Beard [mbphotography.net]

Contents

SLED DRIVER
FLYING THE WORLD'S FASTEST JET
BRIAN SHUL
CENTENNIAL OF FLIGHT LIMITED EDITION

For links to the posts on Hacker News, visit *hackermonthly.com/issue-8*. All articles and comments are reprinted with permission of their original author.
Hacker Monthly is published by Netizens Media and not affiliated with Y Combinator in any way.

Hacker's Guide to Tea

By TONY GEBELY

In addition to caffeine, tea contains an amino acid called L-theanine. "Several studies from Japan and the UK have shown that consumption of 50mg of L-theanine increases alpha wave activity in the brain, with the maximum effect occurring about 80 minutes after consumption. This amount is equivalent to approximately three cups of tea. Alpha waves correspond to a relaxed-but-alert mental state, and are believed to be an important part of selective attention (the ability to choose to pay attention to something and avoid distraction by other stimuli)" [source: teageek.net]. L-theanine in tea produces a type of "mindful awareness" not evident in coffee. This is what prevents the 3pm "coffee crash."

This makes tea an important tool for maintaining mental perspicacity for hours of coding, late night performance, or for getting through those bleak morning hours.

Let's get this out of the way – tea bags suck. Actually, most mainstream tea sucks. Mainstream tea is low quality, blended, and sometimes contains cheap flavorings. There are countless tea shops out there that buy directly from small farmers that produce small crops each season and likely process the tea by hand.

What You Need to Know

All true tea comes from the camellia sinensis plant (photo above). White, Green, Oolong, Yellow, Black, and Pu-erh teas all come from this plant.

Loose tea can be steeped multiple times. Some teas can be re-steeped 20 or more times. The flavor is gradually extracted from the leaves with each subsequent steep.

When shopping for tea, look for companies that offer information about where the tea is from, how it was processed, who grew it, and most importantly—when the tea was harvested.

Steep it

When steeping the tea, be sure the tea can flow freely through the water, this rules out tea bags, tiny tea infusion baskets, tea balls, etc. Ideally, pour water directly over the tea leaves and then strain before drinking. If you must use an infuser, a large finum strainer works nicely and still allows for proper water flow.

Depending on the type of tea you are steeping there are two important variables you must pay attention to: water temperature, and steeping time. I'm assuming you are using good water, as tea is 98% water – using a strong chlorinated water would be a bad idea. In general, hotter water must be used for highly oxidized teas. Remember, you are preparing a drink that you should enjoy, so always take tea instructions with a grain of salt. Experiment often to discover the "sweet spot" with your teas and remember—a good tea is a forgiving tea. If your tea is bitter, reduce the steeping temperature. If your tea is too weak, increase the amount of tea leaves used or increase the steeping time. Here are some guidelines I send out with orders for Chicago Tea Garden:

Tea	Water Temperature	1st Steep	2nd Steep	3rd Steep	4th Steep
White	150-160°F	1 min	1 min	1.5 min	1.75 min
Green	170-180°F	1 min	1 min	1.5 min	1.75 min
Oolong	190-195°F	30 sec	30 sec	45 sec	45 sec
Black	212°F	1 min	1 min	1.5 min	1.5 min
Pu-erh	212°F	30 sec	30 sec	45 sec	1 min

It is not necessary to get real serious about the steeping temperatures, for 195, boil water, take it off the stove, and wait about a minute. For 170, wait longer. Remember, experiment often.

If you want to get serious about steeping your tea, use a yixing pot, or a gaiwan. If you need energy, consider drinking matcha — a suspension of powdered tea. You are actually consuming the leaf so the health benefits and energy received from matcha are greater than from other teas. If you need peace, study the gongfu tea ceremony – it is a great way to relax so you can enjoy and appreciate the tea.

A fresh tea should have a shelf life of approximately two years, a lightly oxidized tea might become stale quicker. Store your tea away from light, heat air, and any strong scents.

Read

There is a lot of good tea information out there. I highly recommend James Norwood Pratt's New Tea Lover's Treasury and Heiss' Story of Tea. If you prefer an online resource, Michael J Coffey has a valuable wiki of his research here [teageek.net/wiki/] and I've assembled a Google Reader bundle of tea blogs [hn.my/teasite]. ■

Tony Gebely is a Chicagoan who has traveled to many tea producing regions and has been studying tea and tea culture for several years. Tony teaches tea courses in Chicago and co-owns Chicago Tea Garden. He also runs World of Tea. If you have any tea related questions he can be found on twitter @WorldofTea.

Reprinted with permission of the original author. First appeared in *hn.my/tea*.
Photographs by Tony Gebely.

Commentary

By JASON FRIED (jasonfried)

A wonderful place to get high quality greens: *www.hibiki-an.com*.

The best hot water kettle with temperature control I've found is: *hn.my/kettle*.

I've tried every kettle and this one is the best. It's all stainless inside too - water never touches plastic.

HACKER JOBS

JS Engineer

GazeHawk *(www.gazehawk.com)*
Mountain View, CA
We're a well-funded YCombinator startup making eye tracking affordable and commonplace. We have a patent-pending technology that lets you run eye tracking studies without custom hardware or software. Now we're looking to augment our team of 3 with another web developer. Read more at *www.gazehawk.com/jobs/*.
To Apply: Email *jobs@gazehawk.com*.

Senior Developer

youDevise, Ltd. *(https://dev.youdevise.com)*
London, England
60-person agile financial software company in London committed to learning and quality (dojos, TDD, continuous integration, exploratory testing). Under 10 revenue-affecting production bugs last year. Release every 2 weeks. Mainly Java, also Groovy, Scala; no prior knowledge of any language needed.
To Apply: Send CV to *jobs@youdevise.com*.

Front-end and Back-end Engineers

Meetup *(www.meetup.com)*
New York
Meetup thinks the world is a better place when groups of people meetup locally, in person, around a common interest. We're reinventing how this is done, but we can't do it alone! We value iterating/launching quickly, pragmatism, and long walks on the beach.
To Apply: *meetup.com/jobs.*

SR-71 Blackbird Communication to Tower

By BRIAN SHUL

Photographs by Marcus Beard [mbphotography.net]

There were a lot of things we couldn't do in an SR-71, but we were the fastest guys on the block and loved reminding our fellow aviators of this fact. People often asked us if, because of this fact, it was fun to fly the jet. Fun would not be the first word I would use to describe flying this plane—intense, maybe, even cerebral. But there was one day in our Sled experience when we would have to say that it was pure fun to be the fastest guys out there, at least for a moment.

It occurred when Walt and I were flying our final training sortie. We needed 100 hours in the jet to complete our training and attain Mission Ready status. Somewhere over Colorado we had passed the century mark. We had made the turn in Arizona and the jet was performing flawlessly. My gauges were wired in the front seat and we were starting to feel pretty good about ourselves, not only because we would soon be flying real missions but because we had gained a great deal of confidence in the plane in the past ten months. Ripping across the barren deserts 80,000 feet below us, I could already see the coast of California from the Arizona border. I was, finally, after many humbling months of simulators and study, ahead of the jet.

I was beginning to feel a bit sorry for Walter in the back seat. There he was, with no really good view of the incredible sights before us, tasked with monitoring four different radios. This was good practice for him for when we began flying real missions, when a priority transmission from headquarters could be vital. It had been difficult, too, for me to relinquish control of the radios, as during my entire flying career I had controlled my own transmissions. But it was part of the division of duties in this plane and I had adjusted to it. I still insisted on talking on the radio while we were on the ground, however. Walt was so good at many things, but he couldn't match my expertise at sounding smooth on the radios, a skill that had been honed sharply with years in fighter squadrons where the slightest radio miscue was grounds for beheading. He understood that and allowed me that luxury. Just to get a sense of what Walt had to contend with, I pulled the radio toggle switches and monitored the frequencies along with him. The predominant radio chatter was from Los Angeles Center, far below us, controlling daily traffic in their sector. While they had us on their scope (albeit briefly), we were in uncontrolled airspace and normally would not talk to them unless we needed to descend into their airspace.

We listened as the shaky voice of a lone Cessna pilot asked Center for a read-out of his ground speed. Center replied: "November Charlie 175, I'm showing you at ninety knots on the ground." Now the thing to understand

Photo: SR-71 Take Off Beale AFB 1985, *http://www.flickr.com/photos/stuseeger/3358525219/*.

about Center controllers, was that whether they were talking to a rookie pilot in a Cessna, or to Air Force One, they always spoke in the exact same, calm, deep, professional tone that made one feel important. I referred to it as the "Houston Center voice." I have always felt that after years of seeing documentaries on this country's space program and listening to the calm and distinct voice of the Houston controllers, that all other controllers since then wanted to sound like that and that they basically did. And it didn't matter what sector of the country we would be flying in, it always seemed like the same guy was talking. Over the years that tone of voice had become somewhat of a comforting sound to pilots everywhere. Conversely, over the years, pilots always wanted to ensure that, when transmitting, they sounded like Chuck Yeager, or at least like John Wayne. Better to die than sound bad on the radios.

Just moments after the Cessna's inquiry, a Twin Beech piped up on frequency, in a rather superior tone, asking for his ground speed in Beech. "I have you at one hundred and twenty-five knots of ground speed." Boy, I thought, the Beechcraft really must think he is dazzling his Cessna brethren.

Then out of the blue, a navy F-18 pilot out of NAS Lemoore came up on frequency. You knew right away it was a Navy jock because he sounded very cool on the radios. "Center, Dusty 52 ground speed check." Before Center could reply, I was thinking to myself, hey, Dusty 52 has a ground speed indicator in that million-dollar cockpit, so why wass he asking Center for a read-out? Then I got it, ol' Dusty there was making sure that every bug smasher from Mount Whitney to the Mojave knows what true speed is. He's the fastest dude in the valley today, and he just wants everyone to know how much fun he is having in his new Hornet. And the reply, always with that same, calm, voice, with more distinct alliteration than emotion: "Dusty 52, Center, we have you at 620 on the ground." And I thought to myself, is this a ripe situation, or what? As my hand instinctively reached for the mic button, I had to remind myself that Walt was in control of the radios. Still, I thought, it must be done—in mere seconds we'll be out of the sector and the opportunity will be lost. That Hornet must die, and die now. I thought about all of our Sim training and how important it was that we developed well as a crew and knew that to jump in on the radios now would destroy the integrity of all that we had worked toward becoming. I was torn.

Somewhere, 13 miles above Arizona, there was a pilot screaming inside his space helmet. Then, I heard it—the click of the mic button from the back seat. That was the very moment that I knew Walter and I had become a crew. Very professionally, and with no emotion, Walter spoke: "Los Angeles Center, Aspen 20, can you give us a ground speed check?" There was no hesitation, and the replay came as if was an everyday request.

"Aspen 20, I show you at one thousand eight hundred and forty-two knots, across the ground." I think it was the forty-two knots that I liked the best, so accurate and proud was Center to deliver that information without hesitation, and you just knew he was smiling. But the precise point at which I knew that Walt and I were going to be really good friends for a long time was when he keyed the mic once again to say, in his most fighter-pilot-like voice: "Ah, Center, much thanks, we're showing closer to nineteen hundred on the money."

For a moment Walter was a god. And we finally heard a little crack in the armor of the Houston Center voice, when L.A. came back with, "Roger that Aspen. Your equipment is probably more accurate than ours. You boys have a good one." It all had lasted for just moments, but in that short, memorable sprint across the southwest, the Navy had been flamed, all mortal airplanes on freq were forced to bow before the King of Speed, and more importantly, Walter and I had crossed the threshold of being a crew. A fine day's work. We never heard another transmission on that frequency all the way to the coast. For just one day, it truly was fun being the fastest guys out there. ■

Brian Shul was an Air Force fighter pilot for 20 years. Shot down in Viet Nam, he spent one year in hospital and was told he'd never fly again. He flew for another 15 years, including the world's fastest jet, the SR-71. As an avid photographer Brian accumulated the world's rarest collection of SR-71 photographs and used them to create the two most popular books on that aircraft, Sled Driver and The Untouchables. Brian today is an avid nature photographer and in high-demand nationwide as a motivational speaker.

Sled Driver Giveaway Challenge

We are giving away a copy of the Limited Edition Sled Driver to one lucky hacker this month. The Limited Edition (picture above) includes Centennial Patch, numbered certificate, presentation book box, and signatures of four prominent Blackbird crew members.

The Challenge

Write a program to show the numerical relationship between "1903", "2003" and "Centennial of Flight".

More Details (and hints!)

1. Submit your code (along with your details) at *hn.my/sled* before 20th January 2011.
2. We will accept your answer in any programming language.
3. Winner will be announced on the next issue (Hacker Monthly #9, February 2011).
4. If there is more than one entry with the correct answer, the winner will be chosen randomly (using Random.org's List Randomizer).
5. Open to all paid subscribers and readers who have purchased a print/ digital copy of issue #8.
6. *Hint*: the answer has nothing to do with 100.
7. No clue yet? Extra hint is hiding somewhere in this issue.
8. Both the latest update and Q&A are available at *hn.my/sled*.

Dear Hacker Monthly Readers,

In 2003, to commemorate the Centenary of the Wright Brothers' first flight, the Limited Edition Sled Driver book was launched. A true collector's item, this lavish remake of the original Sled Driver took the aviation community by storm. Even with the $427 price tag, this one-of-a-kind book has sold steadily for the past seven years and is now on the brink of extinction with just 42 books left.

Recognized worldwide now as the definitive photo essay on the SR-71 Blackbird, this treasured edition has been sent to 39 different countries. It was a huge gamble on our part to print 3800 of these expensive books and they filled our warehouse to overflowing in 2004. While we are thrilled the book has sold so well, it saddens us to now see only a few boxes of this coveted classic in a corner of my office. Few large format aviation books even come close to the longevity of Sled Driver, a book that first appeared on the scene in 1991. I am most proud of the fact that Smithsonian Magazine mentioned that its "Aviation Book of the Year" title was more for the writing, than the prized pictures. The book has now reached cult classic status and is the most quoted book on the Blackbird ever, thanks to the Internet. We added $100 to the price to slow their exodus, but it has done little to discourage those true Blackbird fans out there. We have had the pleasure of meeting so many Sled Heads out there at the various air shows we attended with the book in the past 7 years and have answered countless emails and phone calls weekly. It has been quite a ride.

Sometime in 2011, the final Limited Edition Sled Driver book will depart home-base, and like the plane itself, will be gone forever as an operational entity. There will never be another one quite like it. ■

Brian Shul
Gallery One
15 Dec 2010

Faking It

By SAHIL LAVINGIA

I THINK THAT THE three big areas most start-ups (I use this word loosely) fit into are:

- Providing a product, like 37signals.
- Creating a community, like Pinterest.
- Building a useful platform, like Twilio.

Of course, most successful start-ups end up with many hands in the bucket, but these identify the three main areas pretty well.

The secondary problem that all three encounter, right after building something useful, is generating some sort of scale. The chicken-and-egg situation is a central topic of many, many talks (it certainly was at Start-up School) and for good reason: almost everyone has to deal with it to some extent.

Most of these talks deal with solving the problem. Airbnb says, "Create the supply before you create the demand", and Groupon/Facebook tackled the problem by targeting a certain location and growing. However, I think that the most effective problem is…faking it!

Fake it!

Of course, this doesn't mean to put up false testimonials ("Great app; use daily!" – Barack Obama), create fake real-time activity (extremely easy to spot) or fake your numbers (though I know plenty of start-ups that do, and it works).

Rather, you can engineer your appearance to give off a sense of size. There are plenty of ways of doing this; here are a few I've heard of from famous start-ups.

- Quora's staff started off answering as many questions as they could. This helped create a site that had activity on it, which encouraged other users to participate. Suddenly, they didn't have to spend hours answering questions themselves.
- Whenever Reddit's admin personnel posted submissions, it would randomly generate a submitter's name. This is similar to what Quora did, but slightly more cunning.
- [REDACTED] takes their real-time user numbers and multiplies them by a randomly generated number. Whereas before, it would say "6 users online," it would say "68 users online." They don't fake any activity, but just that extra magnitude of users generates a large amount of actual user activity. Soon, they won't have to do that, as their real user activity will generate enough momentum by itself.
- Several communities, including Pinterest (one I feel especially fond of), just need activity to succeed. It's tough to generate meaningful activity to a point of scale with a small number of employees. To solve this problem, they start as an invite-only community. Because they're invite-only, they have the time to generate useful content and gain really passionate users (when users get invited in when mere mortals aren't, they're more likely to participate, empirically). Then, once you reach a point you're happy with, you can open it up to the public - and bam, massive growth follows.

Creating a presence

I think the most useful thing I've learned over time is to create a presence online, from day one. Create a Facebook account and a Twitter account. Create a website, even if all it does is get you indexed in Google. Have a footer with links to an "about us" page or a "team" page, policy pages, and maybe even a jobs page that says you're not looking for hires at the moment (this link alone changes your project, in my mind, from a side-project to a full-fledged business). Lastly, create a CrunchBase entry about you and your business.

Then, when you want TechCrunch or Mashable to write about you, their Google result (they will, no doubt, search for information about you) will be filled up with pages about you.

Of course, it would be awesome if you had the scale, but most people don't. Build something useful, and then feign the users to get the users. Every start-up starts at 0 users. Every successful start-up uses some sort of social proof to increase their conversion rates. Faking it is probably a harsh term, but it's pretty true, and worthy.

And if it ends up leading to real user activity and a nice valuation, nobody's going to care. ■

Sahil is an 18-year-old USC student soon to be in San Francisco. He enjoys making (useful) stuff for fun and profit, but mostly fun. He's behind Dayta and Color Stream for iPhone; his latest project is Let's Crate.

Reprinted with permission of the original author.
First appeared in *hn.my/fakeit*.

Why I Feel Like a Fraud

By JASON COHEN

"I feel like a fraud. I've been at this for 16 years and I still feel like a fraud. I'm just waiting for the day they see through the façade, but they keep coming back every year." —Jason Young

Ah yes, the awe-inspiring words of confidence from the seasoned entrepreneur. My friend Jason intended this as soothing words of solace during one of my periods of personal freak-out when Smart Bear was in its infancy.

I felt like a fraud every day. Here I was, selling a wobbly, buggy tool and pawning myself off as an expert in a field that didn't exist. My software was the first commercial tool for code review. Every second I felt like I was putting one over on the world.

I would explain how my tool cut code review time in half, but was that actually true or had I just repeated the argument so many times that I stopped questioning it? I would instruct customers on "best practices" for code review, but who am I to tell other people how to critique code? I would orchestrate purchases, but should I be handling large sums of money with no knowledge of accounting, cash-flow, invoicing, purchase orders, or the "enterprise sales" process?

Aren't I too young? Isn't the tool too crappy to charge for? Aren't I too inexperienced? Don't I need an MBA or at least some sales training?

Is Smart Bear a "real company?" What does that even mean?

Objectively, and with hindsight, my feelings were misplaced. The tool really did save time and headache; customers said so. As much as I doubted the title "Code Review Expert," I had developed more experience with more teams in more situations than any one person could (because everyone else was busy doing their actual jobs). And doing sales isn't as mystical and unknowable as I feared.

Still, emotions don't respond to logic. Jason was telling me that these feelings don't go away, even when they ought to.

The other thing he was saying was: You're not alone. As it turns out, it's not even just business founders. Mike Meyers said "I still believe that at any time the No-Talent Police will come and arrest me." Jodie Foster said "I thought it [winning the Oscar] was a fluke. Just the same way as when I walked in the campus at Yale. I thought everybody would find out, and they'd take the Oscar back."

It turns out there's a psycho-babble name for this: Impostor Syndrome. As Inc Magazine points out, studies show that "40% of successful people consider themselves frauds." Ask any small business coach; they'll confirm how prevalent these feelings are. It's even common with PhD candidates.

Although not an official psychological disorder, and generally not crippling, if you have these feelings it's useful to know that it's common and there's something you can do about it.

See if these sound familiar:

- You dismiss compliments, awards, and positive reinforcement as "no big deal."
- You are crushed by mild, constructive criticism.
- You believe you're not as smart/talented/capable as other people think you are.
- You worry others will discover you're not as smart/talented/capable as they think you are.
- You think other people with similar jobs are more "adult" than you are, and they "have their shit together" while you flounder around.
- You feel your successes are due more to luck than ability; with your failures it's the other way around.
- You find it difficult to take credit for your accomplishments.
- You feel that you're the living embodiment of "fake it until you make it."

But wait, how can this be? This overwhelming lack of self-confidence is the opposite of the traditional entrepreneurial stereotype. Don't founders forge ahead even when others say success is impossible? Doesn't a founder invent a new product based on her confidence that others will want it? Doesn't the very idea of starting your own company scream "I'm doing it my way, and my way is better?"

But it does make sense. Consider what it means to be a perfectionist. The perfectionist sees flaws in everyone else's work; there's always a way to make it better - her way. She doesn't respond well to authority dictating how things must be; neither is she comfortable delegating to those who (by her definition) clearly don't care as much as she does.

Sounds like the stereotypical attitude of the arrogant start-up founder, but wait! At the same time, the perfectionist is never happy with her own work either, seeing (inventing?) a never-ending stream of flaws that require attention. No matter how highly others regard her work, the perfectionist insists it's incomplete and unsatisfactory. She can't accept the idea that others would be impressed with her accomplishments, since to her they're mediocre works-in-progress. She worries that one day they'll realize she's right.

Our entrepreneurial motivation is not confidence, it's an insatiable desire to improve. It's not about thinking your ideas are better than everyone else's, but it's about never accepting any idea as being best.

Can these feelings be constructive? Yes, if they're a sign that you're striving to learn and improve. As Andy Wibbels says, "If I don't feel like a fraud at least once a day, then I'm not reaching far enough."

If you aren't scared shitless then why bother?

Here's what it looks like when you're channelling these self-doubts into something constructive:

- I doubt my title as "expert," so every day I read, write, and immerse myself in my field.
- I doubt the quality of my software, so I fix bugs as fast as possible, I write unit tests proactively, and I thank my customers for their patience.
- I doubt I deserve my reputation, so I work hard to earn it.
- I'm not as good as I want to be at speaking/writing/programming/designing/managing, but I can see myself slowly improving.
- I'm not a "real company" yet, so I concentrate on making my customers successful, so they don't care about corporate size or structure.

On the other hand, here's what it looks like when these doubts are harming you:

- I doubt my title as "expert," so every night I worry about what will happen when I'm discovered as a fraud. I'm absent-mindedly looking for trivially-easy jobs I could take where this pressure won't exist. (Looking for an "escape-hatch" is a well-documented behaviour.)
- I doubt the quality of my software, so I spend lots of time covering it up with graphic design and heavy sales pitches.
- I doubt I deserve my reputation, so I live in constant fear of exposure. I can't sleep at night and I loathe myself for lying.
- I'm not as good as I want to be at speaking/writing/programming/designing/managing, so I go out of my way to avoid any of it, and feel like a trapped animal when I'm forced to do it.
- I'm not a "real company" yet, so I feel guilty every time someone gives me money or believes anything I say.

If you're letting these feelings get to you too, at least recognize it so you can deal with it logically. And when logic fails, maybe this will help:

You believe that Mike Meyers and Jodie Foster are talented, right? You might even believe that I'm an expert in peer code review. Yet we doubt ourselves every day. And we're wrong.

You know we're wrong about ourselves; that means you're wrong about yourself too.

Don't stop striving to become better, just stop holding yourself up to an impossible standard.

Sometimes getting it off your chest is the best medicine. ■

Jason is the founder of three companies, all profitable and two exits. He blogs on start-ups and marketing at *blog.ASmartBear.com*.

Commentary

By DANILO CAMPOS (danilocampos)

HOLY SHIT.

I thought this was just me. I spend a lot of time churning on this.

I'm a little shell-shocked at the revelation that others feel this way too. I wish I had something more compelling to contribute than catharsis, but... wow, thanks for (re-) submitting this.

By PATRICK MCKENZIE (patio11)

ME TOO, FOR what it is worth. At least three times this year half my brain was screaming "They're going to find out any second now! Flee, flee!"

It all worked out.

By SUSHAANTU (sushi)

I THINK A PARTICULAR quote from Sh#t my dad says will resonate here.

"That woman was sexy... Out of your league? Son. Let women figure out why they won't screw you, don't do it for them."

Let customers find out if your product is shitty. They won't buy it if it is.

WUFOO

Making web forms

easy + fast + fun

WUFOO

Mortgage Application

Purpose of Mortgage or Loan

Home Loan

WUFOO

Buy a T-Shirt!

Which one do you want? *

Robot Shirt

um

Color *

Black

WUFOO

Customer Satisfaction Survey

Please take a few moments to complete this satisfaction survey.

How long have you used our product / service?

○ Less than a mo

○ 1-6 months

○ 1-3 years

○ Over 3 Years

○ Never used

How often do you

○ Once a week

○ 2 to 3 times a

○ Once a month

○ Less than once

Overall, how satisf

○ Very Satisfied

○ Satisfied

○ Neutral

WUFOO

Join our Mailing List

Stay up to date on the latest news and info!

Your Email Address *

Submit

WUFOO

Workshop Registration

Register now while seats are available!

Name *

Title First Last

Address *

Street Address

Address Line 2

City

Postal / Zip Code

WUFOO

Contact Form

Message *

Life and How to Survive It

I must convey my thanks to the faculty and staff of the Wee Kim Wee School of Communication and Information for inviting me to give you your Convocation speech. It's a wonderful honour and a privilege for me to speak here for ten minutes without fear of contradiction, defamation or retaliation. I say this as a Singaporean and more so, as a husband.

My wife is a wonderful person and perfect in every way, except one. She is the editor of a magazine. She corrects people for a living. She has honed her expert skills over a quarter of a century, mostly by practicing at home during conversations between us.

On the other hand, I am a litigator. Essentially, I spend my day telling people how wrong they are. I make my living, by being disagreeable.

Nevertheless, there is perfect harmony in our matrimonial home. That is because when an editor and a litigator have an argument, the one who triumphs is always the wife.

And so I want to start by giving one piece of advice to the men: when you've already won her heart, you don't need to win every argument.

Marriage is considered to be a great milestone of life. Some of you may already be married. Some of you may never be married. Some of you will get married. Some of you will enjoy the experience of marriage so much that you will be married many, many times. Good for you.

"I'm here to tell you this. Forget about your life expectancy. After all, it's calculated based on an average. And you never, ever want to expect being average."

The next big milestone in your life is today: your graduation. The end of education. You're done learning.

You've probably been told the big lie that "Learning is a lifelong process" and that therefore you will continue studying and taking masters' degrees and doctorates and professorships and so on. You know the sort of people who tell you that? Teachers. Don't you think there is some measure of conflict of interest? They are in the business of learning, after all. Where would they be without you? They need you to be their customers.

The good news is that they're wrong.

The bad news is that you don't need further education because your entire life is over. It is gone. That may come as a shock to some of you. You're in your teens or early twenties. People may tell you that you will live to be 70, 80, or even 90 years old. That is your life expectancy.

I love that term: life expectancy. We all understand the term to mean the average life span of a group of people. But I'm here to talk about a bigger idea, which is what you expect from your life.

You may be very happy to know that Singapore is currently ranked as the country with the third highest life expectancy. We are behind Andorra and Japan, and tied with San Marino. It seems quite clear why people in those countries, and ours, live for so long. We share one thing in common: our football teams are all hopeless. There's very little danger of any of our citizens having their pulses raised by watching us play in the World Cup. Spectators are more likely to be lulled into a gentle and restful nap.

Singaporeans have a life expectancy of 81.8 years. Singapore men live to an average of 79.21 years, while Singapore women live five years longer than that, probably to take into account the additional time they need to spend in the bathroom.

So here you are, in your twenties, thinking that you'll have another 40 years to go. Four decades in which to live long and prosper.

Bad news. Read the papers. There are people dropping dead when they're 50, 40, 30 years old. Or quite possibly just after finishing their Convocation. They would be very disappointed that they didn't meet their life expectancy.

I'm here to tell you this. Forget about your life expectancy.

After all, it's calculated based on an average. And you never, ever want to be average.

Revisit those expectations. You might be looking forward to working, falling in love, marrying, raising a family. You are told that, as graduates, you should expect to find a job paying so much, where your hours are so much, where your responsibilities are so much.

That is what is expected of you. And if you live up to it, it will be an awful waste.

If you expect that, you will be limiting yourself. You will be living your life according to boundaries set by average people. I have nothing against average people. But no one should aspire to be average. And you don't need years of education by the best minds in Singapore to prepare you to be average.

What you should prepare for, is a mess. Life's a mess. You are not entitled to expect anything from it. Life is not fair. Everything does not balance out in the end. Life happens, and you have no control over it. Good and bad things happen to you day by day, hour by hour, and moment by moment. Your degree is a poor armour against fate.

"Do not waste the vast majority of your life doing something you hate so that you can spend the small remaining sliver of your life in modest comfort."

Don't expect anything. Erase all life expectancies. Just live. Your life is over as of today. At this point in time, you have grown as tall as you will ever be, you are physically the fittest you will ever be in your entire life and you are probably looking the best that you will ever look. This is as good as it gets. It is all downhill from here. Or up; No one knows.

What does this mean for you? It is good that your life is over.

Since your life is over, you are free. Let me tell you the many wonderful things that you can do when you are free.

The most important is this: do not work.

Work is anything that you are compelled to do. By its very nature, it is undesirable.

Work kills. The Japanese have a term "Karoshi", which means death from overwork. That's the most dramatic form of how work can kill. But it can also kill you in more subtle ways. If you work, then day by day, bit by bit, your soul is chipped away, disintegrating until there's nothing left. It's like a rock being ground into sand and dust.

There's a common misconception that work is necessary. You will meet people working at miserable jobs. They tell you they are "making a living". No, they're not. They're dying, frittering away their fast-extinguishing lives doing things which are, at best, meaningless and, at worst, harmful.

People will tell you that work ennobles you, and lends you a certain dignity. Work makes you free. The slogan "Arbeit macht frei" was placed at the entrances to a number of Nazi concentration camps. Utter nonsense.

Do not waste the vast majority of your life doing something you hate so that you can spend the small remaining sliver of your life in modest comfort. You may never reach to that end anyway.

Resist the temptation to get a job. Instead, play. Find something you enjoy doing. Do it. Over and over again. You will become good at it for two reasons: you like it, and you do it often. Soon, that will have value in itself.

I like arguing, and I love language. So, I became a litigator. I enjoy it and I would do it for free. If I didn't do that, I would've been in some other type of work that still involved writing fiction - probably a sports journalist.

So what should you do? You will find your own niche. I don't imagine you will need to look very hard. By this time in your life, you will have a very good idea of what you want to do. In fact, I'll go further and say the ideal situation would be that you will not be able to stop yourself from pursuing your passions. By this time you should know what your obsessions are. If you enjoy showing off your knowledge and feeling superior, you might become a teacher.

Find that pursuit that will energize you, consume you and become an obsession. Each day, you must rise with a restless enthusiasm. If you don't, you are working.

Most of you will end up in activities which involve communication. To those of you I have a second message: be wary of the truth. I'm not asking you to speak it, or to write it, for there are times when it is dangerous or impossible to do those things. The truth has a great capacity to offend and injure, and you will find that the closer you are to someone, the more care you must take to disguise or even conceal the truth. Often, there is great virtue in being evasive, or equivocating. There is also great skill. Any child can blurt out the truth, without thought to the consequences. It takes great maturity to appreciate the value of silence.

In order to be wary of the truth, you must first know it. That requires great frankness to yourself. Never fool the person in the mirror.

I have told you that your life is over, that you should not work, and that you should avoid telling the truth. I now say this to you: be hated.

"It is far more easier to find a reason not to love someone, than otherwise. Rejection requires only one reason. Love requires complete acceptance. It is hard work."

It's not as easy as it sounds. Do you know anyone who hates you? Yet every great figure who has contributed to the human race has been hated, not just by one person, but often by a great many. That hatred is so strong that it has caused those great figures to be shunned, abused, murdered and in one famous instance, nailed to a cross.

One does not have to be evil to be hated. In fact, it's often the case that one is hated precisely because one is trying to do right by one's own convictions. It is far too easy to be liked, one merely has to be accommodating and hold no strong convictions. Then one will gravitate towards the centre and settle into the average. That cannot be your role. There are many bad people in the world, and if you are not offending them, you must be bad yourself. Popularity is a sure sign that you are doing something wrong.

The other side of the coin is this: fall in love.

I didn't say "be loved". That requires too much compromise. If one changes one's looks, personality and values, one can be loved by anyone.

Rather, I exhort you to love another human being. I know it may seem odd for me to tell you this. You may expect it to happen naturally, without deliberation. That is false. Modern society is anti-love. We've taken a microscope to everyone to bring out their flaws and shortcomings. It is far more easier to find a reason not to love someone, than otherwise. Rejection requires only one reason. Love requires complete acceptance. It is hard work – the only kind of work that I find palatable.

Loving someone has great benefits. There is admiration, learning, attraction and something which, for want of a better word, we call happiness. In loving someone, we become inspired to better ourselves in every way. We learn the true worthlessness of material things. We celebrate being human. Loving is good for the soul.

Loving someone is therefore very important, and it is also important to choose the right person. Despite popular culture, love doesn't happen by chance, at first sight, across a crowded dance floor. It grows slowly, sinking roots first before branching and blossoming. It is not a silly weed, but a mighty tree that weathers every storm.

You will find that when you have someone to love, that the face is less important than the brain, and the body is less important than the heart.

You will also find that it is no great tragedy if your love is not reciprocated. You are not doing it to be loved back. Its value is to inspire you.

Finally, you will find that there is no half-measure when it comes to loving someone. You either don't, or you do with every cell in your body, completely and utterly, without reservation or apology. It consumes you, and you are reborn, all the better for it.

Don't work. Avoid telling the truth. Be hated. Love someone.

You're going to have a busy life. Thank goodness there's no life expectancy. ■

Adrian Tan is a novelist and a lawyer in Drew & Napier LLC.

Reprinted with permission of the original author. Originally given as a speech to graduating class of 2008 in NTU convocation ceremony. First appeared in *hn.my/life*.

The Myth of the Immortal Hamburger

By J. KENJI LOPEZ-ALT

A FEW WEEKS BACK, I started an experiment designed to prove or disprove whether or not the magic, non-decomposing McDonald's hamburgers that have been making their way around the internet are indeed worthy of disgust or even interest.

By way of introduction, allow myself to quote myself. This is from my previous article:

> *Back in 2008, Karen Hanrahan, of the blog Best of Mother Earth posted a picture of a hamburger that she uses as a prop for a class she teaches on how to help parents keep their children away from junk food... The hamburger she's been using as a prop is the same plain McDonald's hamburger she's been using for what's now going on 14 years. It looks pretty much identical to how it did the day she bought it, and she's not used any means of preservation. The burger travels with her, and sits at room temperature.*
>
> *Now Karen is neither the first nor last to document this very same phenomenon. Artist Sally Davies photographs her 137 day-old hamburger every day for her Happy Meal Art Project. Nonna Joann has chosen to store her happy meal for a year on her blog rather than feed it to her kids. Dozens of other examples exist, and most of them come to the same conclusion: McDonald's hamburgers don't rot.*

The problem with coming to that conclusion, of course, is that if you are a believer in science (and I certainly hope you are!), in order to make a conclusion, you must first start with a few observable premises as a starting point with which you form a theorem, followed by a reasonably rigorous experiment with controls built in place to verify the validity of that theorem.

Thus far, I haven't located a single source that treats this McDonald's hamburger phenomenon in this fashion. Instead, most rely on speculation, specious reasoning, and downright obtuseness to arrive at the conclusion that a McDonald's burger "is a chemical food [, with] absolutely no nutrition."

As I said before, that kind of conclusion is both sensationalistic and specious, and has no place in any of the respectable academic circles in which A Hamburger Today would like to consider itself an upstanding member.

The Theory Behind the Burger

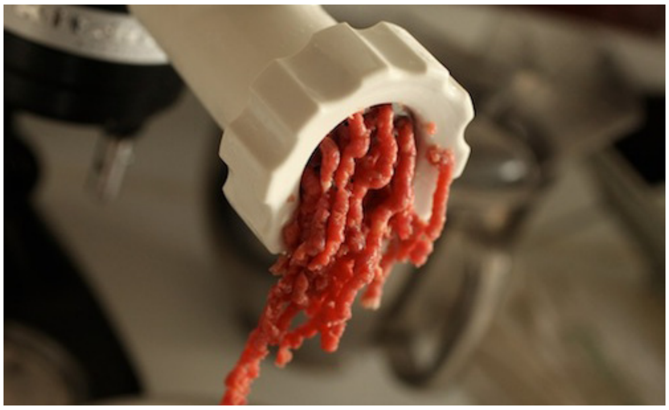
Things we know so far:

1. A plain McDonald's Hamburger, when left out in the open air, does not mold or decompose.
2. In order for mold to grow, a few things need to be present: mold spores, air, moisture, and a reasonably hospitable climate.

Given those two facts, there are a number of theories as to why a McDonald's burger might not rot:

1. There is some kind of chemical preservative in the beef and/or bun and/or the wrapping that is not found in a normal burger and/or bun that creates an inhospitable environment for mold to grow.
2. The high salt level of a McDonald's burger is preventing the burger from rotting.
3. The small size of a McDonald's hamburger is allowing it to dehydrate fast enough that there is not enough moisture present for mold to grow.
4. There are no mold spores present on McDonald's hamburgers, nor in the air in and around where the burgers were stored.
5. There is no air in the environment where the McDonald's hamburgers were stored.

Photographs by J. Kenji Lopez-Alt.



Of these theories, we can immediately eliminate 5, for reasons too obvious to enumerate. As for number 4, it's probably true that there are no live molds on a hamburger when you first receive it, as they are cooked on an extremely hot griddle from both sides to an internal temperature of at least 165°F — hot enough to destroy any mold. But in the air where they were stored? Most likely there's mold present. There's mold everywhere.

Theory 1 is the one most often concluded in the various blogs out there, but there doesn't seem to be strong evidence one way or the other. If we are to believe packaging and nutrition labeling (and I see no reason not to), there are preservatives in a McDonald's bun, but no more than in your average loaf of bread from the supermarket. A regular loaf of supermarket bread certainly rots, so why not the McD's? Their beef is also (according to them) 100% ground beef, so nothing funny going on there, is there?

In order for any test to be considered valid, you need to include a control. Something in which you already know whether or not the variable being tested is present.

In the case of these burgers, that means testing a McDonald's burger against a burger that is absolutely known not to contain anything but beef. The only way to do this, of course, is to cook it myself from natural beef ground at home.

I decided to design a series of tests in order to ascertain the likeliness of each one of these separate scenarios (with the exception of the no-air theory, which frankly, doesn't hold wind—get it?). Here's what I had in mind:

- Sample 1: A plain McDonald's hamburger stored on a plate in the open air outside of its wrapper.
- Sample 2: A plain burger made from home-ground fresh all-natural chuck of the exact dimensions as the McDonald's burger, on a standard store-bought toasted bun.
- Sample 3: A plain burger with a home-ground patty, but a McDonald's bun.
- Sample 4: A plain burger with a McDonald's patty on a store-bought bun.
- Sample 5: A plain McDonald's burger stored in its original packaging.
- Sample 6: A plain McDonald's burger made without any salt, stored in the open air.
- Sample 7: A plain McDonald's Quarter Pounder, stored in the open air.
- Sample 8: A homemade burger the exact dimension of a McDonald's Quarter Pounder.
- Sample 9:A plain McDonald's Angus Third Pounder, stored in the open air.

You may notice that my protocols have been slightly expanded since I first laid them out to you a few weeks ago. That's due to several good ideas in the comments section which I incorporated into my testing the day after the initial publication.

Every day, I monitored the progress of the burgers, weighing each one, and carefully checking for spots of mold growth or other indications of decay. The burgers were left in the open air, but handled only with clean kitchen tools or through clean plastic bags (no direct contact with my hands until the last day).

At this point, it's been 25 days, 23 calm, cool, and collected discussions with my wife about whether that smell in the apartment is coming from the burgers or from the dog, and 16 nights spent sleeping on the couch in the aftermath of those calm, cool, and collected discussions. Aside from my mother, my wife is the fiercest discusser I know.

Frankly, I'm glad this damn experiment is over. On to the results.

“93% of the moisture loss in a regular burger occurs within the first three days.”

The Results

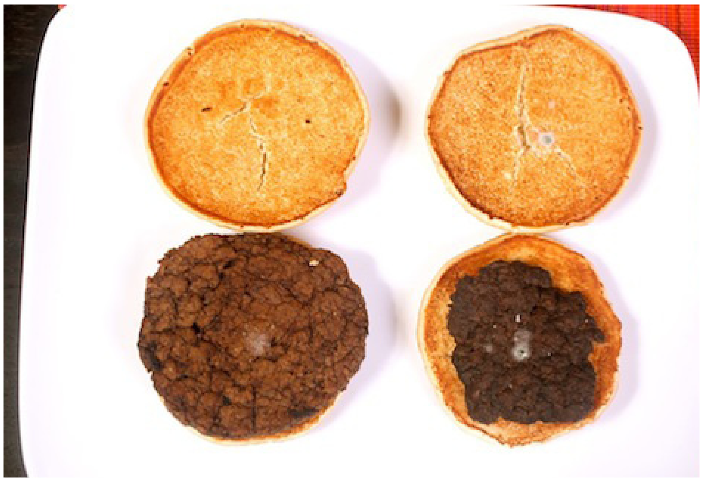
Well, well, well. Turns out that not only did the regular McDonald's burgers not rot, but the home-ground burgers did not rot either. Samples one through five had shrunk a bit (especially the beef patties), but they showed no signs of decomposition. What does this mean?

It means that there's nothing that strange about a McDonald's burger not rotting. Any burger of the same shape will act the same way. The real question is, why?

Well, here's another piece of evidence: Burger number 6, made with no salt, did not rot either, indicating that the salt level has nothing to do with it.

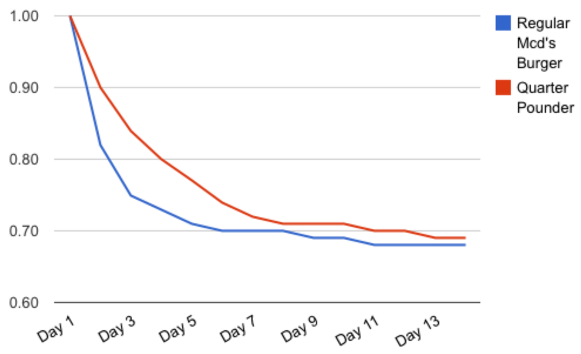
And then we get to the burgers that did show some signs of decay.

Take a look at both the homemade and the McDonald's Quarter Pounder patties:



Very interesting indeed. Sure, there's a slight difference in the actual amount of mold grown, and the homemade patty on the right seems to have shrunk more than the actual Quarter Pounder on the left (I blame that mostly on the way the patties were formed), but on the whole, the results are remarkably similar. That a Quarter Pounder grows mold but a regular-sized McDonald's burger doesn't is some pretty strong evidence in support of Theory 3 from above. Because of the larger size of a Quarter Pounder, it simply takes longer to dehydrate, giving mold more of a chance to grow.

We can prove this by examining the weight charts between the regular burger and the Quarter Pounder. Take a look:



This chart represents the amount of weight lost from the burgers through evaporation every day (both starting weights have been normalized to 1). As you can see, by the end of 2 weeks, both the regular burgers and the Quarter Pounders ended up losing about 31% of their total weight and are pretty much stable. They are essentially burger-jerky. A completely dehydrated product that will never rot, as without moisture, nothing can survive.

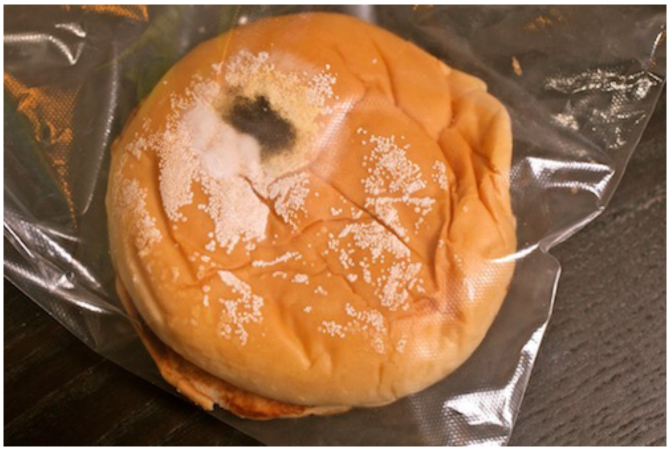
Now the interesting part of the charts is during the first 4 days. As you can see, the blue line representing the regular burger dips down much more precipitously than the red line representing the Quarter Pounder. In fact, 93% of the moisture loss in a regular burger occurs within the first three days, which means that unless mold gets a chance to grow within that time frame, it's pretty much never going to grow.

The Quarter Pounder, on the other hand, takes a full 7 days to dehydrate to the same degree. It's during this extra three day period that the mold growth began to appear (and of course, once the burger had dehydrated sufficiently, the mold growth stopped—the burger looked the same on day 14 as they did on day 7). For the record, the Angus Third Pounder also showed a similar degree of mold growth in the same time frame.

So Can It Mold?

So we've pretty much cleared up all of the confusion, but a keen scientist will notice that one question remains unanswered. We've proven that neither a McDonald's burger nor a regular home-made burger will rot given certain specific conditions, but are there conditions we can create that will cause it to rot, and more importantly, will the McDonald's burger rot as fast as the homemade burger?

The final two burgers I tested were a McDonald's burger and a regular homemade burger of the same dimensions placed in plastic zipper-lock bags side by side. Hopefully the bag would trap in enough moisture. The question: Would they rot?



Indeed they do. Within a week, both burgers were nearly covered in little white spots of mold, eventually turning into the green and black spotted beast you see above.

The Conclusion

So there we have it! Pretty strong evidence in favor of Theory 3: the burger doesn't rot because its small size and relatively large surface area help it to lose moisture very fast. Without moisture, there's no mold or bacterial growth. Of course, that the meat is pretty much sterile to begin with due to the high cooking temperature helps things along as well. It's not really surprising. Humans have known about this phenomenon for thousands of years. After all, how do you think beef jerky is made?

Now don't get me wrong — I don't have a dog in this fight either way. I really couldn't care less whether or not the McDonald's burger rotted or didn't. I don't often eat their burgers, and will continue to not often eat their burgers. My problem is not with McDonald's. My problem is with bad science.

For all of you McDonald's haters out there: Don't worry. There are still plenty of reasons to dislike the company! But for now, I hope you'll have it my way and put aside your beef with their beef. ■

J. Kenji Lopez-Alt [www.goodeater.org/authors/] is a food writer and recipe developer living in New York. He currently resides in Harlem with his wife where he pens the "Food Lab" column for Serious Eats.com [www.seriouseats.com], dedicated to unraveling the mysteries of home cooking with science, soon to be a full-length book published by W. W. Norton. He's the managing editor of Serious Eats, a website dedicated to bringing deliciousness to the world and was recently named one of "40 Big Thinkers Under 40" by Food and Wine magazine.

Commentary

By MICHAEL F BOOTH (mechanical_fish)

I'M UPVOTING THIS partly because it's by J. Kenji (Lopez-)Alt, whom I have regarded as a sort of culinary god ever since I first encountered his awesome piecrust recipe in Cooks Illustrated a few years ago.

(The piecrust is made by substituting vodka for much of the water, which allows the dough to be rolled out without encouraging too much gluten formation and thereby making the crust tough. It is perhaps a shade too much on the crumbly side but makes up for that by being outstandingly tasty, it has now utterly spoiled my taste for the majority of store-bought pies, and it has convinced my friends that I, in turn, am some sort of culinary god, even though this piecrust recipe is idiotically simple, actually simpler than regular piecrust, if such a thing is possible. The lesson here is: Subscribe to Cooks Illustrated and make your loved ones' lives better.)

Obvious to You. Amazing to Others.

By DEREK SIVERS

Any creator of anything knows this feeling:

You experience someone else's innovative work. It's beautiful, brilliant and breath-taking. You're stunned.

Their ideas are unexpected and surprising, but perfect.

You think, "I never would have thought of that. How do they even come up with that? It's genius!"

Afterward, you think, "My ideas are so obvious. I'll never be as inventive as that."

I get this feeling often. Amazing books, music, movies, or even amazing conversations. I'm in awe at how the creator thinks like that. I'm humbled.

But I continue to do my work. I tell my little tales. I share my point of view. Nothing spectacular; just my ordinary thoughts.

One day someone emailed me and said, "I never would have thought of that. How did you even come up with that? It's genius!"

Of course I disagreed, and explained why it was nothing special.

But afterward, I realized something surprisingly profound:

Everybody's ideas seem obvious to them.

I'll bet even John Coltrane or Richard Feynman felt that everything they were playing or saying was pretty obvious.

So maybe what's obvious to me is amazing to someone else?

Hit songwriters, in interviews, often admit that their most successful hit song was one they thought was just stupid, even not worth recording.

We're clearly bad judges of our own creations. We should just put it out and let the world decide.

Are you holding back something that seems too obvious to share? ■

Derek Sivers founded a music distribution company, CD Baby, in 1997, a web hosting company, Hostbaby, in 2000, and sold both in 2008. Since then he's been a popular speaker at the TED Conferences, and writing short essays at *sivers.org*. He's now setting up a few new companies to help musicians.

Scalable PHP Hosting, made easy with the CatN Platform.

Instantly deploy your apps on our super cluster. No code changes necessary. SSH Access, Cron jobs, Git & SVN support all as standard.

from £5 per month

www.catn.com

Why Python Rocks for Research

By HOYT KOEPKE

THE FOLLOWING IS an account of my own experience with Python. Because that experience was so positive, this is an unabashed attempt to promote the use of Python for general scientific research and development. About four years ago, I dropped MATLAB in favor of Python as my primary language for coding research projects. This article is a personal account of how rewarding I found that experience to be.

As I describe in the next sections, the variety and quality of Python's features have spoiled me. Even in small scripts, I now rely on Python's numerous data structures, classes, nested functions, iterators, the flexible function calling syntax, an extensive kitchen-sink-included standard library, great scientific libraries, and outstanding documentation.

To clarify, I am not advocating *just* Python as the perfect scientific programming environment; I am advocating Python plus a handful of mature 3rd-party open source libraries, namely Numpy/Scipy for numerical operations, Cython for low-level optimization, IPython for interactive work, and MatPlotLib for plotting. Later, I describe these and others in more detail, but I introduce these four here so I can weave discussion of them throughout this article.

Given these libraries, many features in MATLAB that enable one to quickly write code for machine learning and artificial intelligence – my primary area of research – are essentially a small subset of those found in Python. After a day learning Python, I was able to still use most of the matrix tricks I had learned in MATLAB, but also utilize more powerful data structures and design patterns when needed.

Holistic Language Design

I once believed that the perfect language for research was one that allowed concise and direct translation from notepad scriblings to code. On the surface, this is reasonable. The more barriers between generating ideas and trying them out, the slower research progresses. In other words, the less one has to think about the actual coding, the better. I now believe, however, that this attitude is misguided.

MATLAB's language design is focused on matrix and linear algebra operations; for turning such equations into one-liners, it is pretty much unsurpassed. However, move beyond these operations and it often becomes an exercise in frustration. R is beautiful for interactive data analysis, and its open library of statistical packages is amazing. However, the language design can be unnatural, and even maddening, for larger development projects. While Mathematica is perfect for interactive work with pure math, it is not intended for general purpose coding.

The problem with the "perfect match" approach is that you lose generalizability very quickly. When the criteria for language design is too narrow, you inevitably choose excellence for one application over greatness for many. This is why universities have graduate programs in computer language design — navigating the pros and cons of various design decisions is extremely difficult to get right. The extensive use of Python in everything from system administration and website design to numerical number-crunching shows that it has, indeed, hit the sweet spot. In fact, I've anecdotally observed that becoming better at R leads to skill at interacting with data, becoming better at MATLAB leads to skill at quick-and-dirty scripting, but becoming better at Python leads to genuine programming skill.

Practically, in my line of work, the downside is that some matrix operators that are expressable using syntactical constructs in MATLAB become function calls (e.g. `x = solve(A, y)` instead of `x = A \ y`). In exchange for this extra verbosity — which I have not found problematic — one gains incredible flexibility and a language that is natural for everything from automating system processes to scientific research. The coder doesn't have to switch to another language when writing non-scientific code, and allows one to easily leverage other libraries (e.g. databases) for scientific research.

Furthermore, Python allows one to easily leverage object oriented and functional design patterns. Just as different problems call for different ways of thinking, so also different problems call for different programming paradigms. There is no doubt that a linear, procedural style is natural for many scientific problems. However, an object oriented style that builds on classes having internal functionality and external behavior is a perfect design pattern for others. For this, classes in Python are full-featured and practical. Functional programming, which builds on the power of iterators and functions-as-variables, makes many programming solutions concise and intuitive. Brilliantly, in Python, everything can be passed around as an object, including functions, class definitions, and modules. Iterators are a key language component and Python comes with a full-featured iterator library. While it doesn't go as far in any of these categories as flagship paradigm languages such as Java or Haskell, it does allow one to use some very practical tools from these paradigms. These features combine to make the language very flexible for problem solving, one key reason for its popularity.

Readability

To reiterate a recurrent point, Python's syntax is *very* well thought out. Unlike many scripting languages (e.g. Perl), readability was a primary consideration when Python's syntax was designed. In fact, the joke is that turning pseudocode into correct Python code is a matter of correct indentation.

This readability has a number of beneficial effects. Guido van Rossum, Python's original author, writes:

> *This emphasis on readability is no accident. As an object-oriented language, Python aims to encourage the creation of reusable code. Even if we all wrote perfect documentation all of the time, code can hardly be considered reusable if it's not readable. Many of Python's features, in addition to its use of indentation, conspire to make Python code highly readable.*

In addition, I've found it encourages collaboration, and not just by lowering the barrier to contributing to an open source Python project. If you can easily discuss your code with others in your office, the result can be better code and better coders.

As two examples of this, consider the following code snippet:

```
def classify(values, boundary=0):
  "Classifies values as being below (False) or above (True)
a boundary."
  return [(True if v > boundary else False) for v in
values]

# Call the above function
classify(my_values, boundary=0.5)
```

Let me list three aspects of this code. First, it is a small, self-contained function that only requires three lines to define, including documentation (the string following the function). Second, a default argument for the boundary is specified in a way that is instantly readable (and yes, that does show up when using Sphinx for automatic documentation). Third, the list processing syntax is designed to be readable. Even if you are not used to reading Python code, it is easy to parse this code — a new list is defined and returned from the list `values` using `True` if a particular value `v` is above `boundary` and `False` otherwise. Finally, when calling functions, Python allows named arguments — this universally promotes clarity and reduces stupid bookkeeping bugs, particularly with functions requiring more than one or two arguments.

Permit me to contrast these features with MATLAB. With MATLAB, globally available functions are put in separate files, discouraging the use of smaller functions and — in practice — often promotes cut-and-paste programming, the bane of debugging. Default arguments are a pain, requiring conditional coding to set unspecified arguments. Finally, specifying arguments by name when calling is not an option, though one popular but artificial construct — alternating names and values in an argument list — allows this to some extent.

Balance of High Level and Low Level Programming

The ease of balancing high-level programming with low-level optimization is a particular strong point of Python code. Python code is meant to be as high level as reasonable — I've heard that in writing similar algorithms, on average you would write six lines of C/C++ code for every line of Python. However, as with most high-level languages, you often sacrifice code speed for programming speed.

One sensible approach around this is to deal with higher level objects — such as matrices and arrays — and optimize operations on these objects to make the program acceptably fast. This is MATLAB's approach and is one of the keys to its success; it is also natural with Python. In this context, speeding code up means vectorizing your algorithm to work with arrays of numbers instead of with single numbers, thus reducing the overhead of the language when array operations are optimized.

Abstractions such as these are absolutely essential for good scientific coding. Focusing on higher-level operations over higher-level data types generally leads to massive gains in coding speed and coding accuracy. Python's extension type system seamlessly allows libraries to be designed around this idea. Numpy's array type is a great example.

However, existing abstractions are not always enough when you're developing new algorithms or coding up new ideas. For example, vectorizing code through the use of arrays is powerful but limited. In many cases, operations really need loops, recursion, or other coding structures that are extremely efficient in optimized, compiled machine code but are not in most interpreted languages. As variables in many interpreted languages are not statically typed, the code can't easily be compiled into optimized machine code. In the scientific context, Cython provides the perfect balance between the two by allowing either.

Cython works by first translating Python code into equivalent C code that runs the Python interpreted through the Python C API. It then uses a C compiler to create a shared library that can be loaded as a Python module. Generally, this module is functionally equivalent to the original Python module and usually runs marginally faster. The advantage, however, is that Cython allows one to statically type variables — e.g. `cdef int i` declares `i` to be an integer. This gives massive speedups, as typed variables are now treated using low-level types rather than Python variables. With these annotations, your "Python" code can be as fast as C — while requiring very little actual knowledge of C.

Practically, a few type declarations can give you incredible speedups. For example, suppose you have the following code:

```
def foo(A):
  for i in range(A.shape[0]):
    for j in range(A.shape[1]):
      A[i,j] += i*j
```

where `A` is a 2d NumPy array. This code uses interpreted loops and thus runs fairly slowly. However, add type information and use Cython:

```
def cyfoo(ndarray[double, ndim=2] A):
  cdef size_t i, j

  for i in range(A.shape[0]):
    for j in range(A.shape[1]):
      A[i,j] += i*j
```

Cython translates necessary Python operations into calls to the Python C-API, but the looping and array indexing operations are turned into low level C code. For a 1000 x 1000 array, on my 2.4 GHz laptop, the Python version takes 1.67 seconds, while the Cython version takes only 3.67 milliseconds (a vectorized version of the above using an outer product took 15.1 ms).

A general rule of thumb is that your program spends 80% of its time running 20% of the code. Thus a good strategy for efficient coding is to write everything, profile your code, and optimize the parts that need it. Python's profilers are great, and Cython allows you to do the latter step with *minimal* effort.

Language Interoperability

As a side affect of its universality, Python excels at gluing other languages together. One can call MATLAB functions from Python (through the MATLAB engine) using MLabWrap, easing transitions from MATLAB to Python. Need to use that linear regression package in R? RPy puts it at your fingertips. Have fast FORTRAN code for a particular numerical algorithm? F2py will effortless generate a wrapper. Have general C or C++ libraries you want to call? Ctypes, Cython, or SWIG are three ways to easily interface to it (my favorite is Cython). Now, if only all these were two way streets...

Documentation System

Brilliantly, Python incorporates module, class, function, and method documentation directly into the language itself. In essence, there are two levels of comments — programming level comments (start with #) that are ignored by the compiler, and documentation comments that are specified by a doc string after the function or method name. These documentation strings add tags to the methods which are accessible by anyone using an interactive Python shell or by automatic documentation generators.

The beauty of Python's system becomes apparent when using Sphinx, a documentation generation system originally built for Python language documentation. To allow sufficient presentation flexibility, it allows reStructuredText directives, a simple, readable markup language that is becoming widely used in code documentation. Sphinx works easily with embedded doc-strings, but it is useful beyond documentation — for example, my personal website, my course webpages when I teach, my code documentation sites, and, of course, Python's main website are generated using Sphinx.

One helpful feature for scientific programming is the ability to put LaTeX equations and plots directly in code documentation. For example, if you write:

```
.. math:: \Gamma(z) = \int_0^\infty x^{z-1}e^x\,dx
```

in the doc string, it is rendered in the webpage as

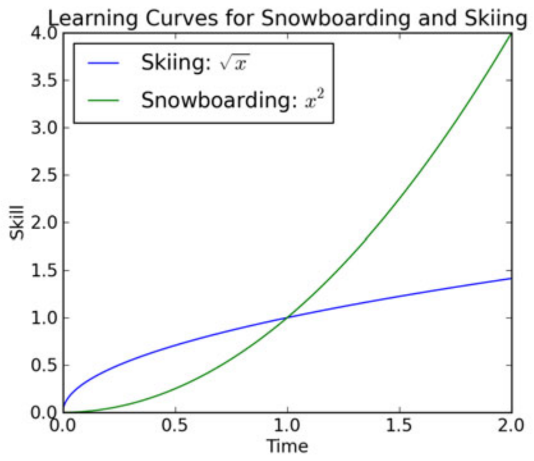
$$\Gamma(z) = \int_0^\infty x^{z-1}e^x\,dx$$

Including plots is easy. The following doc-string code:

```
.. plot::
  import numpy as np
  import matplotlib.pyplot as plt

  x = np.linspace(0,2, 1000)
  plt.figure() plt.plot(x, np.sqrt(x), label = r"Skiing: $\
  sqrt{x}$")
  plt.plot(x, x**2, label = r"Snowboarding: $x^2$")
  plt.title("Learning Curves for Snowboarding and Skiing")
  plt.xlabel("Time") ;
  plt.ylabel("Skill") plt.legend(loc='upper left')
  plt.show()
```

gives



In essence, this enables not only comments about the code, but also comments about the science and research behind your code, to be interwoven into the coding file.

Hierarchical Module System

Python uses modular programming, a popular system that naturally organizes functions and classes into hierarchical namespaces. Each Python file defines a module. Classes, functions, or variables that are defined in or imported into that file show up in that module's namespace. Importing a module either creates a local dictionary holding that module's objects, pulls some of the module's objects into the local namespace. For example, `import hashlib` binds `hashlib.md5` to hashlib's md5 checksum function; alternately, `from hashlib import md5` binds `md5` to this function. This helps programming namespaces to follow a hierarchical organization.

On the coding end, a Python file defines a module. Similarly, a directory containing an `__init__.py` Python file is treated the same way, files in that directory can define submodules, and so on. Thus the code is arranged in a hierarchical structure for both the programmer and the user.

Permit me a short rant about MATLAB to help illustrate why this is a great feature. In MATLAB, all functions are declared in the global namespace, with names determined by filenames in the current path variable. However, this discourages code reusability by making the programmer do extra work keeping disparate program components separate. In other words, without a hierarchical structure to the program, it's difficult to extract and reuse specific functionality. Second, programmers must either give their functions long names, essentially doing what a decent hierarchical system inherently does, or risk namespace conflicts which can be difficult to resolve and result in subtle errors. While this may help one to throw something together quickly, it is a horrible system from a programming language perspective.

Data Structures

Good programming requires having and using the correct data structures for your algorithm. This is almost universally underemphasized in research-oriented coding. While proof-of-concept code often doesn't need optimal data structures, such code causes problems when used in production. This often — though it's rarely stated or known explicitly — limits the scalability of a lot of existing code. Furthermore, when such features are not natural in a language's design, coders often avoid them and fail to learn and use good design patterns.

Python has lists, tuples, sets, dictionaries, strings, threadsafe queues, and many other types built-in. Lists hold arbitrary data objects and can be sliced, indexed, joined, split, and used as stacks. Sets hold unordered, unique items. Dictionaries map from a unique key to anything and form the real basis of the language. Heaps are available as operations on top of lists (similar to the C++ STL heaps). Add in NumPy, and one has an n-dimensional array structure that supports optimized and flexible broadcasting and matrix operations. Add in SciPy, and you have sparse matrices, kd-trees, image objects, time-series, and more.

Available Libraries

Python has an *impressive* standard library packaged with the program. Its philosophy is "batteries-included", and a standard Python distribution comes with built-in database functionality, a variety of data persistence features, routines for interfacing with the operating system, website interfacing, email and networking tools, data compression support, cryptography, xml support, regular expressions, unit testing, multithreading, and much more. In short, if I want to take a break from writing a bunch of matrix manipulation code and automate an operating system task, I don't have to switch languages.

Numerous libraries provide the needed functionality for scientific . The following is a list of the ones I use regularly and find to be well-tested and mature:

- **NumPy/SciPy:** This pair of libraries provide array and matrix structures, linear algebra routines, numerical optimization, random number generation, statistics routines, differential equation modeling, Fourier transforms and signal processing, image processing, sparse and masked arrays, spatial computation, and numerous other mathematical routines. Together, they cover most of MATLAB's basic functionality and parts of many of the toolkits, and include support for reading and writing MATLAB files. Additionally, they now have great documentation (vastly improved from a few years ago) and a very active community.
- **IPython:** One of the best things in Python is IPython, an enhanced interactive Python shell that makes debugging, profiling code, interactive plotting. It supports tab completion on objects, integrated debugging, module finding, and more — essentially, it does almost everything you'd expect a command line programming interface to do. Additionally,
- **Cython:** Referenced earlier, Cython is a painless way of embedding compiled, optimized bits of code in a larger Python program.
- **SQLAlchemy:** SQLAlchemy makes leveraging the power of a database incredibly simple and intuitive. It is essentially a wrapper around an SQL database. You build queries using intuitive operators, then it generates the SQL, queries the database, and returns an iterator over the results. Combining it with sqlite — embedded in Python's standard library — allows one to leverage databases for scientific work with impressive ease. And, if you tell sqlite to build its database in memory, you've got another powerful data structure. To slightly plagiarize xkcd, SQLAlchemy makes databases fun again.
- **PyTables:** PyTables is a great way of managing large amounts of data in an organized, reliable, and efficient fashion. It optimizes resources, automatically transferring data between disk and memory as needed. It also supports on-the-fly (DE)compression and works seamlessly with NumPy arrays.
- **PyQt:** For writing user interfaces in C++, I recommend it is, in my experience, difficult to beat QT. PyQt brings the ease of QT to Python. And I do mean ease — using the interactive QT designer, I've build a reasonably complex GUI-driven scientific application with only a few dozen lines of custom GUI code. The

entire thing was done in a few days. The code is cross-platform over Linux, Mac OS X, and Windows. If you need to develop a front end to your data framework, and don't mind the license (GPL for PyQT, LGPL for QT), this is, in my experience, the easiest way to do so.

- **TreeDict:** Without proper foresight and planning, larger research projects are particularly prone to the second law of thermodynamics: over time, the organization of parameters, options, data, and results becomes increasingly random. TreeDict is a Python data structure I designed to fight this. It stores hierarchical collections of parameters, variables, or data, and supports splicing, joining, copying, hashing, and other operations over tree structures. The hierarchical structure promotes organization that naturally tracks the conceptual divisions in the program — for example, a single file can define all parameters while reflecting the structure of the rest of the code.
- **Sage:** Sage doesn't really fit on this list as it packages many of the above packages into a single framework for mathematical research. It aims to be a complete solution to scientific programming, and it incorporates over a hundred open source scientific libraries. It builds on these with a great notebook concept that can really streamline the thought process and help organize general research. As an added bonus, it has an online interface for trying it out. As a complete package, I recommend newcomers to scientific Python programming try Sage first; it does a great job of unifying available tools in a consistent presentation.
- **Enthought Python Distribution:** Also packaging these many libraries into a complete package for scientific computing, the Enthought Python Distribution is distributed by a company that contributes heavily to developing and maintaining these libraries. While there are commercial support options, it is free for academic use.

Testing Framework

I do not feel comfortable releasing code without an accompanying suite of tests. This attitude, of course, reflects practical programmer wisdom; code that is guaranteed to function a certain way — as encapsulated in these unit tests — is reusable and dependable. While packaging test code without does not always equate with code quality, there is a strong correlation. Unfortunately, the research community does not often emphasize writing proper test code, due partly to that emphasis being directed, understandably, towards technique, theory, and publication. But this is exactly why a no-boilerplate, practical and solid testing framework and simple testing constructs like assert statements are so important. Python provides a built-in, low barrier-to-entry testing framework that encourages good test coverage by making the fastest workflow, including debugging time, involve writing test cases. In this way, Python again distinguishes itself from its competitors for scientific code.

Downsides

No persuasive essay is complete without an honest presentation of the counterpoints, and indeed several can be made here. In fact, many of my arguments invite a counterargument — with so many options available at every corner, where does one start? Having to make decisions at each turn could paralyze productivity. For most applications, wouldn't a language with a rigid but usually adequate style — like MATLAB — be better?

While one can certainly use a no-flair scripting style in Python, I agree with this argument, at least to a certain extent. However, the situation is not uniformly bad — rather, it's a bit like learning to ski versus learning to snowboard. The first day or two learning to snowboard is always horrid, while one can pick up basic skiing quite quickly. However, fast-forward a few weeks, and while the snowboarder is perfecting impressive tricks, the skier is still working on not doing the splits. An exaggerated analogy, perhaps, but the principle still holds: investment in Python yields impressive rewards, but be prepared for a small investment in learning to leverage its power.

The other downside with using Python for general scientific coding is the current landscape of conventions and available resources. Since MATLAB is so common in many fields, it is often conventional to publish open research code in MATLAB (except in some areas of mathematics, where Python is more common on account of Sage; or in statistics, where R is the lingua franca). While MLabWrap makes this fairly workable, it does means that a Python programmer may need to work with both languages and possess a MATLAB license. Anyone considering a switch should be aware of this potential inconvenience; however, there seems to be a strong movement within scientific research towards Python — largely for the reasons outlined here.

A Complete Programming Solution

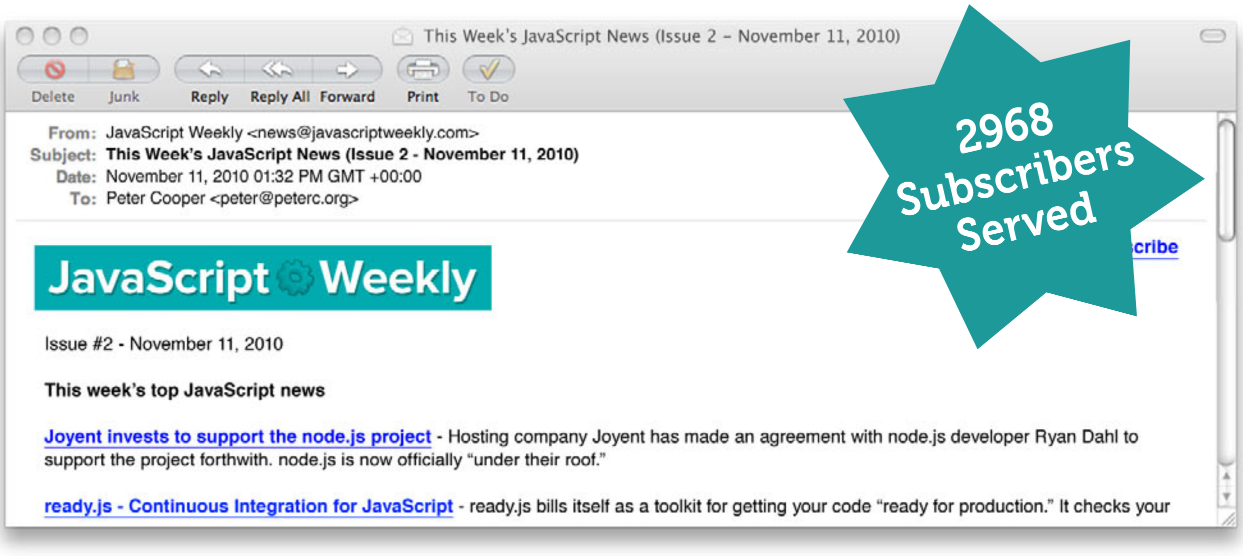
In summary, and reiterating my point that Python is a complete programming solution, I mention three additional points, each of which would make a great final thought. First, it is open source and completely free, even for commercial use, as are many of the key scientific libraries. Second, it runs natively on Windows, Mac OS, linux, and others, as does its standard library and the third party libraries I've mentioned here. Third, it fits quick scripting and large development projects equally well. A quick perusal of some success stories on Python's website showcases the diversity of environments in which Python provides a scalable, well-supported, and complete programming solution for research and scientific coding. However, the best closing thought is due to Randall Monroe, the author of xkcd: "Programming is fun again!" ■

Hoyt Koepke is a PhD student in the Statistics Department at the University of Washington studying optimization, ranking models, probability theory, and machine learning/artificial intelligence. As a teen, he learned to program when his parents would only let him play computer games he wrote himself, and subsequently got a MSc in computer science from the University of British Columbia following a BA in physics at the University of Colorado. He can be contacted at *hoytak@stat.washington.edu* or visited online at *www.stat.washington.edu/~hoytak*.

JavaScriptWeekly.com

A concise, once-weekly roundup of **JavaScript** news to your Inbox.

You can filter out topics you don't want too.



From: JavaScript Weekly <news@javascriptweekly.com>
Subject: This Week's JavaScript News (Issue 2 - November 11, 2010)
Date: November 11, 2010 01:32 PM GMT +00:00
To: Peter Cooper <peter@peterc.org>

JavaScript Weekly

Issue #2 - November 11, 2010

This week's top JavaScript news

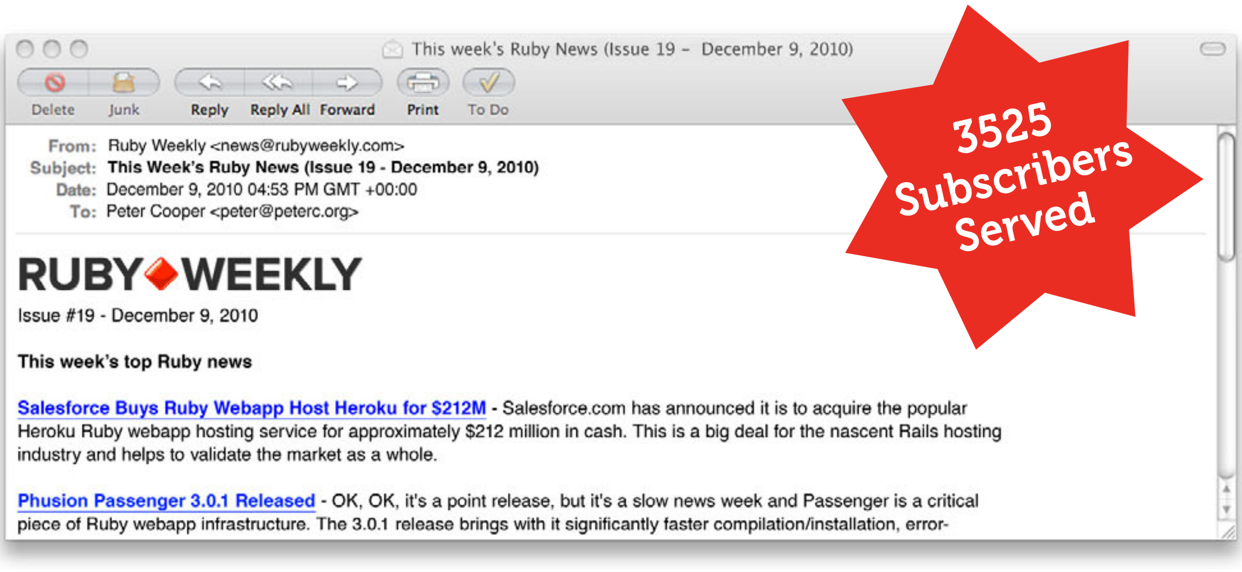
Joyent invests to support the node.js project - Hosting company Joyent has made an agreement with node.js developer Ryan Dahl to support the project forthwith. node.js is now officially "under their roof."

ready.js - Continuous Integration for JavaScript - ready.js bills itself as a toolkit for getting your code "ready for production." It checks your

OR...

RubyWeekly.com

A concise, once-weekly roundup of **Ruby** and **Rails** news to your Inbox.



From: Ruby Weekly <news@rubyweekly.com>
Subject: This Week's Ruby News (Issue 19 - December 9, 2010)
Date: December 9, 2010 04:53 PM GMT +00:00
To: Peter Cooper <peter@peterc.org>

RUBY WEEKLY

Issue #19 - December 9, 2010

This week's top Ruby news

Salesforce Buys Ruby Webapp Host Heroku for $212M - Salesforce.com has announced it is to acquire the popular Heroku Ruby webapp hosting service for approximately $212 million in cash. This is a big deal for the nascent Rails hosting industry and helps to validate the market as a whole.

Phusion Passenger 3.0.1 Released - OK, OK, it's a point release, but it's a slow news week and Passenger is a critical piece of Ruby webapp infrastructure. The 3.0.1 release brings with it significantly faster compilation/installation, error-

Cracking Passwords in the Cloud

Amazon's New EC2 GPU Instances

By THOMAS ROTH

As of today, Amazon EC2 is providing what they call "Cluster GPU Instances": An instance is the Amazon cloud that provides you with the power of two NVIDIA Tesla "Fermi" M2050 GPUs. The exact specifications are as follows:

22 GB of memory
33.5 EC2 Compute Units (2 x Intel Xeon X5570, quad-core "Nehalem" architecture)
2 x NVIDIA Tesla "Fermi" M2050 GPUs
1690 GB of instance storage
64-bit platform
I/O Performance: Very High (10 Gigabit Ethernet)
API name: cg1.4xlarge

GPUs are known to be the best hardware accelerators for cracking passwords, so I decided to give it a try and try to find out how fast this instance type can be used to crack SHA1 hashes. Using the CUDA-Multiforcer, I was able to crack all hashes from a file with a password length of 1-6 characters in only 49 minutes (1 hour costs $2.10, by the way).

```
The compute done was:
Reference time: 2950.1 seconds
Stepping rate: 249.2M MD4/s
Search rate: 3488.4M NTLM/s
```

One more time, this just shows that SHA1 for password hashing is deprecated - You really don't want to use it anymore! It would be better to use something like scrypt or PBKDF2! Just imagine a whole cluster of these machines cracking passwords for you, which is now easy for anybody to do, thanks to Amazon! They're providing a pretty comfortable and large scale password cracking facility for everybody!

Installation Instructions

I used the "Cluster Instances HVM CentOS 5.5 (AMI Id: ami-aa30c7c3)" machine image provided by Amazon , since it was the only one with built-in CUDA support, and selected "Cluster GPU (cg1.4xlarge, 22GB)" as the instance type. After launching the instance and SSHing into it, you can continue by installing the cracker:

I decided to install the "CUDA-Multiforcer" version 0.7 as it's the latest version, and the source code is also available. To compile it, you first need to download the "GPU Computing SDK code samples".

```
# wget http://developer.download.nvidia.com/compute/
  cuda/3_2/sdk/gpucomputingsdk_3.2.12_linux.run
# chmod +x gpucomputingsdk_3.2.12_linux.run
# ./gpucomputingsdk_3.2.12_linux.run
(Just press enter when asked for the installation directory
and the CUDA directory).
```

Now we need to install the g++ compiler:

```
# yum install automake autoconf gcc-c++
```

The next step is compiling the libraries of the SDK samples:

```
# cd ~/NVIDIA_GPU_Computing_SDK/C/
# make lib/libcutil.so
# make shared/libshrutil.so
```

Now it's time to download and compile the CUDA-Multiforcer:

```
# cd ~/NVIDIA_GPU_Computing_SDK/C/
# wget http://www.cryptohaze.com/releases/CUDA-Multiforcer-
src-0.7.tar.bz2 -O src/CUDA-Multiforcer.tar.bz2
# cd src/
# tar xjf CUDA-Multiforcer.tar.bz2
# cd CUDA-Multiforcer-Release/argtable2-9/
# ./configure && make && make install
# cd ../
```

As the Makefile of the CUDA-Multiforcer doesn't work out of the box, just open it up and find the line:

```
CCFILES := -largtable2 -lcuda
```

Replace "CCFILES" with "LINKFLAGS" so that the line looks like this:

```
LINKFLAGS := -largtable2 -lcuda
```

And type make. If everything worked out, you should have a file: "~/NVIDIA_GPU_Computing_SDK/C/bin/linux/release/CUDA-Multiforcer" now. You can try the Multiforcer by doing something like this:

```
# export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
# export LD_LIBRARY_PATH=/usr/local/cuda/lib64:$LD_LIBRARY_
  PATH
# cd ~/NVIDIA_GPU_Computing_SDK/C/src/CUDA-Multiforcer-
  Release/
# ../../bin/linux/release/CUDA-Multiforcer -h SHA1 -f test_
  hashes/Hashes-SHA1-Full.txt --min=1 --max=6 -c charsets/
  charset-upper-lower-numeric-symbol-95.chr
```

Congratulations, you now have a fully working CUDA-based hash-cracker running on an Amazon EC2 instance.

Getting the Facts Straight

At this point, I have to get some facts straight: What I did was benchmark the speed of the new instance type for cracking SHA1 hashes. My first result was that it took 49 minutes to do a "95 characters, 6 digit long" brute force attack on a list of 14 hashes. The thing that was new is that due to the new Amazon offering, everyone is able to spawn a 100 or mode node cluster in the cloud and distribute the task of cracking passwords onto these nodes, especially since cracking hashes is perfectly suitable for massive parallelization! An attacker would be able to spawn a gigantic cluster of nodes using some stolen credit card information and it would be no problem for him to crack an 8 character long password within a nice time frame.

The reason I said that SHA1 is deprecated for storing passwords is easy to explain: SHA1 was never made to store passwords. SHA1 is a hash algorithm; it was made for verifying data. It was made to be as fast and as collision free as possible, and that's the problem when using it for storing passwords: It's too fast! The speed of computers is increasing incredibly fast, and so brute forcing will get faster and faster, and the new cloud offerings make parallelization of such use tasks easy and affordable. Instead of hash algorithms, one should use Key Derivation Functions like PBKDF2 or scrypt. Some of these functions hash passwords thousands of times and make brute forcing them a lot harder.

I hope that this article helps people to understand the real impact of using the cloud for cracking passwords. ■

Thomas Roth is a consultant for security and software engineering from Germany whose main interests are exploiting techniques, low-level programming languages and cryptographic algorithms. He started implementing and optimizing hash algorithms like MD5 and SHA1 on GPUs, using the CUDA and the OpenCL framework recently. Some of his private work can be found on his blog [stacksmashing.net] or on Twitter @stacksmashing.

Code Iceberg

By GABRIEL WEINBERG

A LOT OF GOOD products have features that appear somewhat trivial to replicate, but in reality would be quite complex to do so. I call these features code icebergs because they expose what a casual observer or competitor imagines is a weekend hackathon, but underneath there is a humongous mass of necessarily complicated code that makes everything work as seamlessly as it appears.

In my experience, the iceberg part of a code iceberg often involves handling of a lot edge cases. These edge cases are sometimes actually created by making the user interface simpler, e.g. less or free-form input fields.

At my current startup, DuckDuckGo, a good example is the seemingly straightforward task of taking Wikipedia and turning it into good Zero-click Info to display against queries. At first blush it's trivial — I mean come on, the Wikipedia dumps output something called abstract.xml with a description of "extracted abstracts for Yahoo."

Yet when you get into it and start exposing it to real users, you surface all those edge cases. That dump in particular is actually completely unusable IMHO and I ended up discarding it within a few days of discovering it. It chokes on lots of things.

Wikipedia has templates, disambiguation pages, initial warnings and infoboxes, redirects, malformed/complicated sentences, etc. etc., all of which you want to deal with if you don't want glaring errors. And then once you're in there, you might as well start capturing more good stuff like related topics, categories, the right images, good external links, etc. etc. And what about updating it in real time? It starts to really add up.

I like code icebergs. They're really a marvel to look at when you can see the whole picture. They also lure competitors in, who often get sunk (at least initially) not understanding the scope of the problem. They're good barriers to entry, fuel in build vs buy decisions, and the underpinnings of good UX. ■

Gabriel Weinberg is the founder of Duck Duck Go, a search engine. He is also an active angel investor, based out of Valley Forge, PA. More info at his homepage: *ye.gg*.

Photo: Danmark O, Fohn Fjord, Renodde.70°N/26°W, *http://www.flickr.com/photos/rietje/76566707/*.

Commentary

By SDR (sdrinf)

CODE ICEBERG IS in the eye of the beholder. Recently started bizdev-people consistently underestimate the time requirements for certain well-exercised tasks.

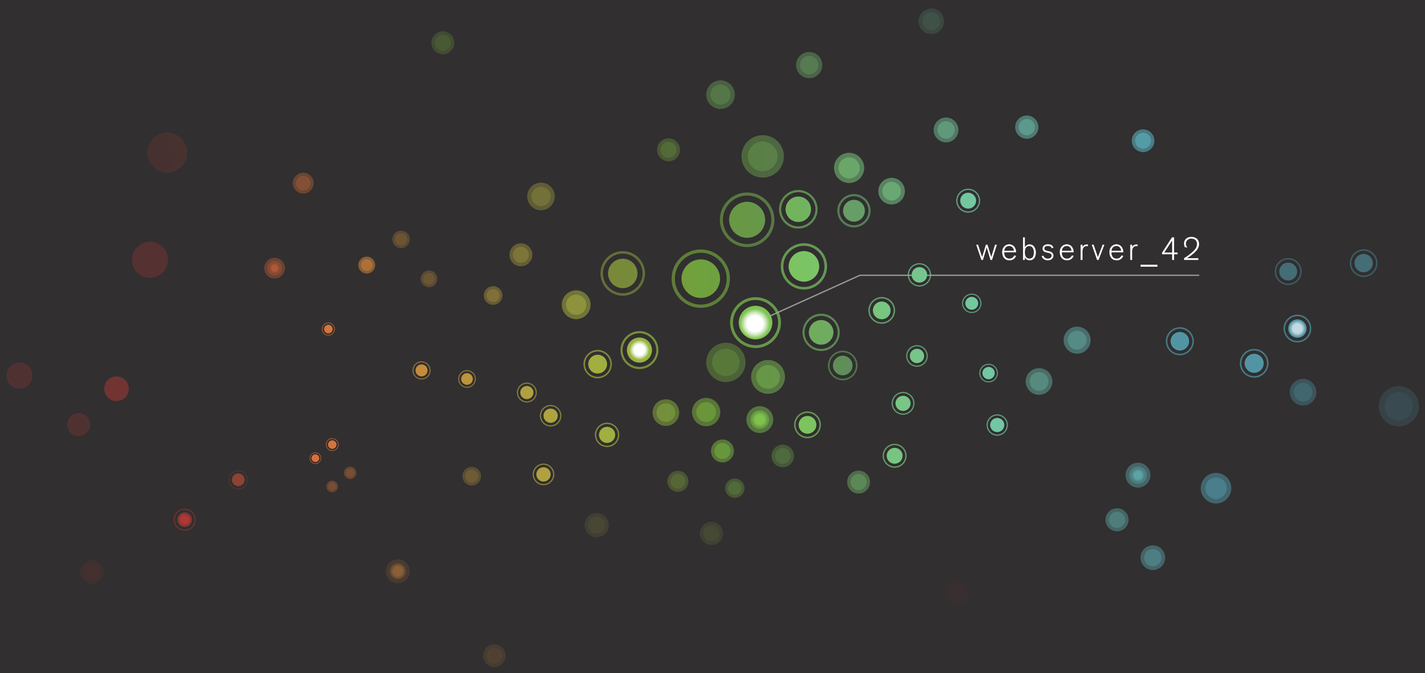
Some of the most common icebergs are:

- Form validation (seriously -one of the most highly exercised user-interaction paths; it's all over the place, and scales semi-exponentially with the number of fields).
- Search ("how hard could it be? you just put an input form there, then figure out what the user thought, then display it" -exact quote).
- Anything that has to process natural language. I mean everything. Wanna split up a text into sentences? How do you differentiate between dr. mr., 2004. jun. , and valid sentence-enders? Generating a definite article ("a", "an") before a noun? Keep in mind that 1,2,@,$,=, and other characters might also be valid noun first-letters :) etc.

In my experience, the best anti-iceberg pattern is to follow a portfolio approach, and for each requirements which smells like iceberg, have a fallback plan in place -ie. after *N* hours of sunken investment, execution shifts to plan B. Usually works out much better, than banging away on the same problem for days.

These are your servers

These are your servers on Cloudkick



Any questions?

cloudkick.com
415.779.5425

support for 8 clouds + dedicated hardware

cloudkick
the best way to manage the cloud

Google and Microsoft Cheat on Slow-Start. Should You?

By BEN STRONG

A Quest for Speed

I decided a couple of weeks ago that I wanted to build an app, most likely a web app. Being a premature optimizer by nature, my first order of business (after deciding I needed to learn to draw) was to find the absolute fastest way to serve up a web page. The Google home page is the fastest loading page I know of, so I thought a good place to start would be to figure out how they do it and then replicate their strategy.

The full story of my search is below, but the short version is that to match Google's page load times you have to cheat on the TCP Slow-Start algorithm. It appears that stretching the parameters a little bit is fairly common, but Google and Microsoft push it a lot further than most. This may well be common knowledge in web development circles, but it was news to me.

Some Sleuthing

My first step was to measure the load time of www.google.com over my home cable-modem connection. As a first pass, I timed the download with curl:

```
$ time curl www.google.com > /dev/null
% Total % Received % Xferd Average Speed Time Time Time
Current Dload Upload Total Spent Left Speed
100 8885 0 8885 0 0 115k 0 -:-:- -:-:- -:-:- 173k
real 0m0.085s
```

Holy smokes, that was fast! We were able to open a TCP connection, make an HTTP request, receive an 8KB response, and close the connection, all in 85ms! That's even faster than I expected, and demonstrates that it should be possible to build an app with a page-load time below the threshold that humans perceive as instantaneous (about 150ms, according to one study). Sign me up.

Curious about how they pulled that off (did someone sneak into my house and install a GGC node in the attic?), I fired up tcpdump to take a closer look. What I saw surprised me:

```
$ tcpdump -ttttt host www.google.com

# 3-way handshake (RTT 16ms)
00:00:00.000000 IP 192.168.1.21.52238 > 74.125.227.16
00:00:00.016255 IP 74.125.227.16.http > 192.168.1.21.522380
0:00:00.016376 IP 192.168.1.21.52238 > 74.125.227.16

# client sends request and server acks
00:00:00.017437 IP 192.168.1.21.52238 > 74.125.227.16
00:00:00.037139 IP 74.125.227.16.http > 192.168.1.21.52238

# server sends 8 segments in the space of 3ms (interspersed
with client acks)
00:00:00.067151 IP 74.125.227.16.http > 192.168.1.21.52238
00:00:00.069693 IP 74.125.227.16.http > 192.168.1.21.52238
00:00:00.069814 IP 192.168.1.21.52238 > 74.125.227.16
[...]
00:00:00.070990 IP 192.168.1.21.52238 > 74.125.227.16

# connection close (RTT 22 ms)
00:00:00.071300 IP 192.168.1.21.52238 > 74.125.227.16
00:00:00.093299 IP 74.125.227.16.http > 192.168.1.21.52238
00:00:00.093469 IP 192.168.1.21.52238 > 74.125.227.16
```

On the performance front, this is really exciting. They actually managed to deliver the whole response in just 70ms, 30ms of which was spent generating the response (come on Google, you can do better than 30ms). That means that a load time under 50ms should be possible.

How they accomplished that is what surprised me. The rate at which a server can send data over a new connection is limited by the TCP Slow-Start algorithm, which works as follows: The server maintains a congestion window which controls how many TCP segments it can send before receiving ACKs from the client. The server starts with a small initial window (IW), and then for each ACK received from the client increases the window size by one segment until it either reaches the client's receive window size or encounters congestion. This allows the server to discover the true bandwidth of the path in a way that's fair to other flows and minimizes congestion.

If you look at the trace, though, you'll notice that the server is actually sending the entire 8 segment response before there's time for the first client ACK to reach it. This is a clear violation of RFC-3390, which defines the following algorithm for determining the max IW:

```
The upper bound for the initial window is given more pre-
cisely in, (1): min (4*MSS, max (2*MSS, 4380 bytes))

Note: Sending a 1500 byte packet indicates a maximum
segment size (MSS) of 1460 bytes (assuming no IP or TCP
options). Therefore, limiting the initial window's MSS to
4380 bytes allows the sender to transmit three segments
initially in the common case when using 1500 byte packets.
```

www.google.com is indeed advertising an MSS of 1460, allowing it an IW of 3 segments according to the RFC. In our trace, they appear to be using an IW of at least 8, which allows them to shave off 2 round trips (~50ms) over an IW of 3 for this request. This raises the question: just how far will they go? Let's request a larger file and see what happens:

```
$ tcpdump -i en1 -ttttt host www.google.com

# 3-way handshake (RTT 22ms)
00:00:00.000000 IP 192.168.1.21.52287 > 74.125.227.50
00:00:00.022780 IP 74.125.227.50.http > 192.168.1.21.52287
00:00:00.022913 IP 192.168.1.21.52287 > 74.125.227.50

# client request and server ack
00:00:00.023699 IP 192.168.1.21.52287 > 74.125.227.5
000:00:00.048205 IP 74.125.227.50.http > 192.168.1.21.52287

# server sends 9 segments in 4ms (interspersed with client
acks)
00:00:00.082766 IP 74.125.227.50.http > 192.168.1.21.52287
00:00:00.083077 IP 74.125.227.50.http > 192.168.1.21.52287
00:00:00.083118 IP 192.168.1.21.52287 > 74.125.227.50
[...]
00:00:00.086836 IP 192.168.1.21.52287 > 74.125.227.50

# 24ms after first client ack was sent, we get 2 more segments
00:00:00.107116 IP 74.125.227.50.http > 192.168.1.21.52287
00:00:00.107403 IP 74.125.227.50.http > 192.168.1.21.52287
00:00:00.107518 IP 192.168.1.21.52287 > 74.125.227.50

#connection close (RTT 25ms)
00:00:00.107938 IP 192.168.1.21.52287 > 74.125.227.50
00:00:00.129947 IP 74.125.227.50.http > 192.168.1.21.52287
00:00:00.130071 IP 192.168.1.21.52287 > 74.125.227.50
```

Interestingly, the server waits for ~1 RTT after sending 9 segments, indicating an IW of 9. This suggests that the value was tuned for the home page (or for the similarly-sized search results page).

How Common is This?

So, is this common practice that I've just never noticed before, or is Google the only one doing it? I thought I'd run traces against a few more sites and try to deduce their IWs. Here's what I found:

Akamai:	4
Amazon:	3
Cisco:	2
Facebook:	4
Limelight Networks:	4
Yahoo:	3

It looks like goosing the IW to 4 is pretty common practice, but I was about to give up on finding anyone pushing as far as Google until, almost as an afterthought, I tried www.microsoft.com. You have to see it to believe it:

```
$ tcpdump -i en1 -ttttt host www.microsoft.com

# 3-way handshake (RTT 92ms)
00:00:00.000000 IP 192.168.1.21.52625 > wwwco1vip.microsoft.com
00:00:00.091960 IP wwwco1vip.microsoft.com. > 192.168.1.21.52625
00:00:00.092094 IP 192.168.1.21.52625 > wwwco1vip.microsoft.com

# request form client and server ack
00:00:00.092909 IP 192.168.1.21.52625 > wwwco1vip.microsoft.com
00:00:00.189451 IP wwwco1vip.microsoft.com > 192.168.1.21.52625

#server sends 43 segments without pause, for a total of
almost 64KB!!! (the full client receive window size)
00:00:00.189780 IP wwwco1vip.microsoft.com > 192.168.1.21.52625
00:00:00.190009 IP wwwco1vip.microsoft.com > 192.168.1.21.52625
00:00:00.190055 IP 192.168.1.21.52625 > wwwco1vip.microsoft.com
[...]
00:00:00.210471 IP wwwco1vip.microsoft.com > 192.168.1.21.52625

finally, the server waits for an ACK before continuing
00:00:00.282291 IP wwwco1vip.microsoft.com > 192.168.1.21.52625
00:00:00.282420 IP 192.168.1.21.52625 > wwwco1vip.microsoft.com
[...]
```

Microsoft appears to be skipping Slow-Start altogether and setting the IW to the full client receive buffer size. Crazy!

Some Discussion

A search for "google TCP initial window" turns up a Google-authored research paper and Internet-Draft proposing a change to the Slow-Start algorithm to allow an IW of up to 10 segments (IW10). Interesting.

There's also a lively ongoing discussion on the IETF TMRG mailing list. I haven't read every post (there have been hundreds over the last few months), but it seems that most of the participants are approaching this as a theoretical problem, not as an issue that is actually occurring in the wild and needs to be addressed. The Google engineers on the mailing list have taken on a more frustrated tone recently, so it's possible that they decided the best way to make forward progress was to just turn it on and see whether the Internet actually melts down. It's also possible that I happen to be part of an ongoing test that they're running.

I wasn't able to find any discussion relevant to what I saw in my Microsoft trace.

Conclusions

Fast is good. I'm excited to see that sub-100ms page loads are possible, and it's a shame to not be able to take full advantage of modern networks because of protocol limitations (http being the limiting protocol, btw).

Being non-standards-compliant in a way that privileges their flows relative to others seems more than a little hypocritical from a company that's making such a fuss about network neutrality.

I'm not really qualified to render judgment on whether IW10 is a net positive, but after reading the discussion (and considering that the internet hasn't actually melted down), I'm inclined to think that it is.

I'm pretty confident that turning off slow-start entirely, as Microsoft seems to be doing in my trace, is a very bad thing (maybe even a bug).

So, this leaves the question, what should I do in my app (and what should you do in yours)? Join the arms race or sit on the sidelines and let Google have all the page-load glory? I'll let you know what I decide[1]. ■

Notes

1. Read the follow-up here: *hn.my/slowstart2*.

Ben Strong is a software architect and entrepreneur living in Austin, TX. He founded Bluelark Systems and SugarSync, and was most recently Principal Architect at Palm. He is currently starting a new venture.

Commentary

By TOM HUGHES-CROUCHER (sh1mmer)

THIS ISN'T MUCH of a secret. As it says in the article Google are lobbying to change the initial window size in the RFC. A lot of people here at Yahoo! want to see that too, and personally I think we should be more aggressive with our initial window, RFC be damned.

This topic was covered really well by Amazon's John Rauser at Velocity Conf.

To address the points in the conclusion:

1. Fast is good. Fast is also profit.
2. The net-neutrality argument here is totally bogus, anyone that knows how can up their slow-start window today if they choose to. There doesn't really have anything to do with traffic shaping.
3. Google have been using their usual data driven approach to support their proposal for IETF. We need a lot more of that. It's great. The only way we can really find out how the Internet in general will react to changes like this is to test them in some real world environment.
4. I agree, slow-start is a good algorithm with a very valid purpose. The real problem here is that the magic numbers powering it aren't being kept inline with changes to connectivity technology and increases in consumer/commercial bandwidth.

Why You Should Know Just A Little AWK

By GREG GROTHAUS

In grad school, I once saw a professor I was working with grab a text file and in seconds manipulate it into little pieces so deftly it blew my mind. I immediately decided it was time for me to learn AWK, which he had so clearly mastered.

To this day, 90% of the programmers I talk to have never used AWK. Knowing 10% of AWK's already small syntax, which you can pick up in just a few minutes, will dramatically increase your ability to quickly manipulate data in text files. Below I'll teach you the most useful stuff - not the "fundamentals", but 5 minutes worth of practical information that will get you the most of what I think is interesting in this little language.

AWK is a fun little programming language. It is designed for processing input strings. A (different) professor once asked my networking class to implement code that would take a spec for an RPC service and generate stubs for the client and the server. This professor made the mistake of telling us we could implement this in any language. I decided to write the generator in AWK, mostly as an excuse to learn more AWK. It surprised me because the code ended up much shorter and much simpler than it would have been in any other language I've ever used (Python, C++, Java, ...). There is enough to learn about AWK to fill half a book, and I've read that book, but you're unlikely to be writing a full-fledged spec parser in AWK. Instead, you just want to do things like find all of your log lines that come from ip addresses whose components sum up to 666, for kicks and grins. Read on!

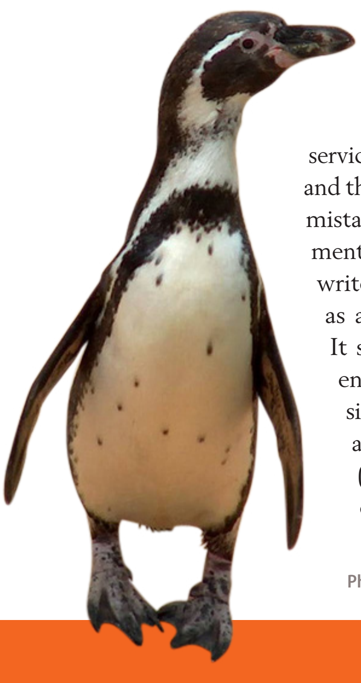


Photo: Pingouin, *http://www.flickr.com/photos/52345210@N08/4816336371/*.

For our examples, assume we have a little file (`logs.txt`) that looks like the one below. If it wraps in your browser, this is just 2 lines of logs each staring with an ip address.

```
07.46.199.184 [28/Sep/2010:04:08:20] "GET /robots.txt
HTTP/1.1" 200 0 "msnbot"
123.125.71.19 [28/Sep/2010:04:20:11] "GET / HTTP/1.1" 304
-  "Baiduspider"
```

These are just two log records generated by Apache, slightly simplified, showing Bing and Baidu wandering around on my site yesterday.

AWK works like anything else (ie: `grep`) on the command line. It reads from stdin and writes to stdout. It's easy to pipe stuff in and out of it. The command line syntax you care about is just the command AWK followed by a string that contains your program.

```
awk '{print $0}'
```

Most AWK programs will start with a "{" and end with a "}". Everything in between there gets run once on each line of input. Most AWK programs will print something. The program above will print the entire line that it just read, `print` appends a newline for free. `$0` is the entire line. So this program is an identity operation - it copies the input to the output without changing it.

AWK parses the line in to fields for you automatically, using any whitespace (space, tab) as a delimiter, merging consecutive delimiters. Those fields are available to you as the variables $1, $2, $3, etc.

```
echo 'this is a test' | awk '{print $3}'  // prints 'a'
awk '{print $1}' logs.txt
```

Output:

```
07.46.199.184
123.125.71.19
```

Easy so far, and already useful. Sometimes I need to print from the end of the string though instead. The special variable, `NF`, contains the number of fields in the current line. I can print the last field by printing the field `$NF` or I can just manipulate that value to identify a field based on it's position from the last. I can also print multiple values simultaneously in the same `print` statement.

```
echo 'this is a test' | awk '{print $NF}'  // prints "test"
awk '{print $1, $(NF-2) }' logs.txt
```

Output:

```
07.46.199.184 200
123.125.71.19 304
```

More progress - you can see how, in moments, you could strip this log file to just the fields you are interested in.

Another cool variable is `NR`, which is the row number being currently processed. While demonstrating `NR`, let me also show you how to format a little bit of output using `print`. Commas between arguments in a `print` statement put spaces between them, but I can leave out the comma and no spaces are inserted.

```
awk '{print NR ") " $1 " -> " $(NF-2)}' logs.txt
```

Output:

```
1) 07.46.199.184 -> 200
2) 123.125.71.19 -> 304
```

Powerful, but nothing hard yet, I hope. By the way, there is also a `printf` function that works much the way you'd expect if you prefer that form of formatting. Now, not all files have fields that are separated with whitespace. Let's look at the date field:

```
$ awk '{print $2}' logs.txt
```

Output:

```
[28/Sep/2010:04:08:20]
[28/Sep/2010:04:20:11]
```

The date field is separated by "/" and ":" characters. I can do the following within one AWK program, but I want to teach you simple things that you can string together using more familiar unix piping because it's quicker to pick up a small syntax. What I'm going to do is pipe the output of the above command through another AWK program that splits on the colon. To do this, my second program needs two {} components. I don't want to go into what these mean, just to show you how to use them for splitting on a different delimiter.

```
$ awk '{print $2}' logs.txt | awk 'BEGIN{FS=":"}{print $1}'
```

Output:

```
[28/Sep/2010
[28/Sep/2010
```

I just specified that I wanted a different FS (field separator) of ":" and that I wanted to then print the first field. No more time, just dates! The simplest way to get rid of that prefix [character is with sed, which you are likely already familiar with:

```
$ awk '{print $2}' logs.txt  | awk 'BEGIN{FS=":"}{print
$1}' | sed 's/\[//'
```

Output:

```
28/Sep/2010
28/Sep/2010
```

I can further split this on the "/" character if I want to by using the exact same trick, but I think you get the point. Next, lets learn just a tiny bit of logic. If I want to return only the 200 status lines, I could use grep, but I might end up with an ip address that contains 200, or a date from year 2000. I could first grab the 200 field with AWK and then grep, but then I lose the whole line's context. AWK supports basic if statements. Lets see how I might use one:

```
$ awk '{if ($(NF-2) == "200") {print $0}}' logs.txt
```

Output:

```
07.46.199.184 [28/Sep/2010:04:08:20] "GET /robots.txt
HTTP/1.1" 200 0 "msnbot"
```

There we go, returning only the lines (in this case only one) with a 200 status. The `if` syntax should be very familiar and require no explanation. Let me finish up by showing you one simple example of AWK code that maintains state across multiple lines. Lets say I want to sum up all of the status fields in this file. I can't think of a reason I'd want to do this for statuses in a log file, but it makes a lot of sense in other cases like summing up the total bytes returned across all of the logs in a day or something. To do this, I just create a variable which automatically will persist across multiple lines:

```
$ awk '{a+=$(NF-2); print "Total so far:", a}' logs.txt
```

Output:

```
Total so far: 200
Total so far: 504
```

Nothing doing. Obviously in most cases, I'm not interested in cumulative values but only the final value. I can, of course, just use `tail -n1`, but I can also print stuff after processing the final line using an END clause:

```
$ awk '{a+=$(NF-2)}END{print "Total:", a}' logs.txt
```

Output:

```
Total: 504
```

If you want to read more about AWK, there are several good books and plenty of online references. You can learn just about everything there is to know about AWK in a day with some time to spare. Getting used to it is a bit more of a challenge as it really is a little bit different way to code - you are essentially writing only the inner part of a for loop. Come to think of it, this is a lot like how MapReduce feels, which is also initially disorienting. ■

Greg Grothaus is a software engineer at Google working on Search Quality, where he is responsible for maintaining the quality of the search results in Google's main search index. Prior to Google, he studied Bioinformatics at Virginia Tech. Visit his blog at *gregable.com*.

Commentary

By JON PINCUS (jdp23)

BACK IN THE 80s I wrote a 500-line program analysis tool in AWK. One day the woman I was going out with handed me a printout I had left at her place, saying something along the lines of "here's your AWK code". She wasn't a programmer so I was stunned that she knew it was AWK, and very impressed too.

Years later I ran into Brian Kernighan at a conference and told him the story, ending it with "and that's when I knew she was the woman for me." He looked at me like I was nuts.

By KLIMENT YANEV (Kliment)

AWK IS A great and oft-forgotten tool. Not only is it useful, the AWK way of thinking about stream processing generalizes nicely to a bunch of other areas. You have a block that runs before anything else happens, a block run just before the program exits, and a block run for every piece of input. In AWK, the input is a line of text, but nothing stops you from generalizing this to say a frame from a video (split into channels in various colorspaces, fed through a processing pipeline, returning another, processed image), a sound frame, a sensor measurement...

HACKER JOBS

Front-end Developer

Adioso *(adioso.com)*
Global (Company based in Australia & SF)
Join our quest to redefine travel search. Code your way around the world! We're seeking passionate travellers who also happen to be masters of HTML, CSS, JavaScript and jQuery, ideally with some knowledge of Python/Django & PostgreSQL.
For more info visit: *blog.adioso.com/workadioso-front-end-developer*.
To Apply: Email *work@adioso.com*.

Front-end and Backend Hackers

BackType (*www.backtype.com*)
San Francisco
"Fantastic opportunity to join a fast growing startup with a great team" — Y Combinator (Investor)
"The @backtype guys are killing it. Join them and change the world while getting rich" — Chris Sacca (Investor)
"@backtype is one of the few startups that reminds me of Palantir early-stage" — Stephen Cohen (Founder of Palantir Technologies)
To Apply: *www.backtype.com/jobs.*

Web Engineer

Ginzamarkets, Inc. (*ginzametrics.com*)
Mountain View, CA
Ginzamarkets, Inc. produces Ginzametrics, a new kind of marketing platform for advertisers and agencies. We focus on SEO with plans to expand into a $40B+ market. You'll have the chance to build a platform that solves big problems for almost every company doing serious marketing online.
To Apply: Email *hackers@ginzametrics.com*.

Sled Driver Giveaway Challenge Extra Hint: "If you add up the individual numbers in the years 1903 and 2003, it equals 18, the number of letters needed to write Centennial of Flight." (source: www.sleddriver.com/patch.html)

Dream. Design. Print.

MagCloud, the revolutionary new self-publishing web service by HP, is changing the way ideas, stories, and images find their way into peoples' hands in a printed magazine format.

HP MagCloud capitalizes on the digital revolution, creating a web-based marketplace where traditional media companies, upstart magazine publishers, students, photographers, designers, and businesses can affordably turn their targeted content into print and digital magazine formats.

Simply upload a PDF of your content, set your selling price, and HP MagCloud takes care of the rest—processing payments, printing magazines on demand, and shipping orders to locations around the world. All magazine formatted publications are printed to order using HP Indigo technology, so they not only look fantastic but there's no waste or overruns, reducing the impact on the environment.

Become part of the future of magazine publishing today at www.magcloud.com.

25% Off the First Issue You Publish

Enter promo code **HACKER** when you set your magazine price during the publishing process.

Coupon code valid through February 28, 2011.
Please contact promo@magcloud.com with any questions.

MAGCLOUD