



Growing Tomatoes

Joe Hewitt

HACKERMONTHLY

Issue 39 August 2013

Curator

Lim Cheng Soon

Contributors

Patrick Smith
Joe Hewitt
Jon Wheatley
Brennan Dunn
Ian Langworth
ridiculous_fish
Gergely Kalman
Chris Parnin
Maroun Najjar

Illustrators

Parko Polo
Ben O'Brien

Proofreaders

Emily Griffin
Sigmarie Soto

Ebook Conversion

Ashish Kumar Jha

Printer

MagCloud

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

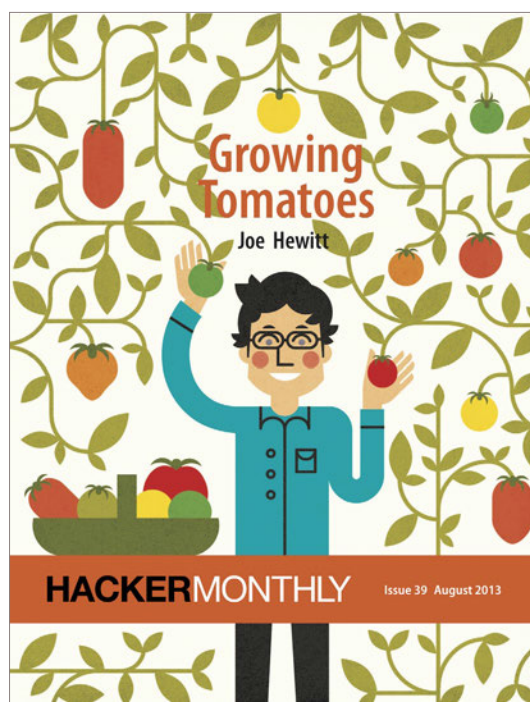
ads@hackermonthly.com

Contact

contact@hackermonthly.com

Published by

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



Cover Illustration: Parko Polo

Contents

FEATURES

05 **The Exploding Toilet**

By PATRICK SMITH

10 **Growing Tomatoes**

By JOE HEWITT

STARTUPS

14 **Making A Physical Product**

By JON WHEATLEY

18 **The Freelancer's Guide To Recurring Revenue**

By BRENNAN DUNN

PROGRAMMING

22 **Vim After 11 Years**

By IAN LANGWORTH

26 **Yahoo! Chat — A Eulogy**

By RIDICULOUS_FISH

28 **How I Made Porn 20x More Efficient with Python**

By GERGELY KALMAN

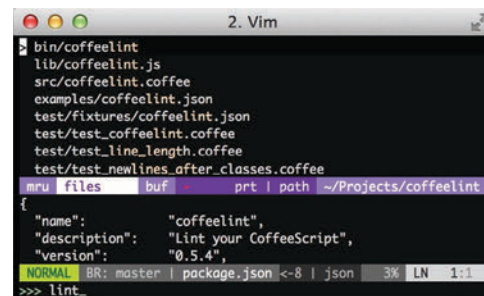
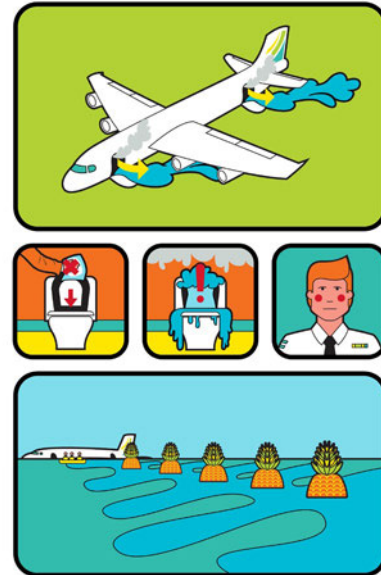
32 **Programmer, Interrupted**

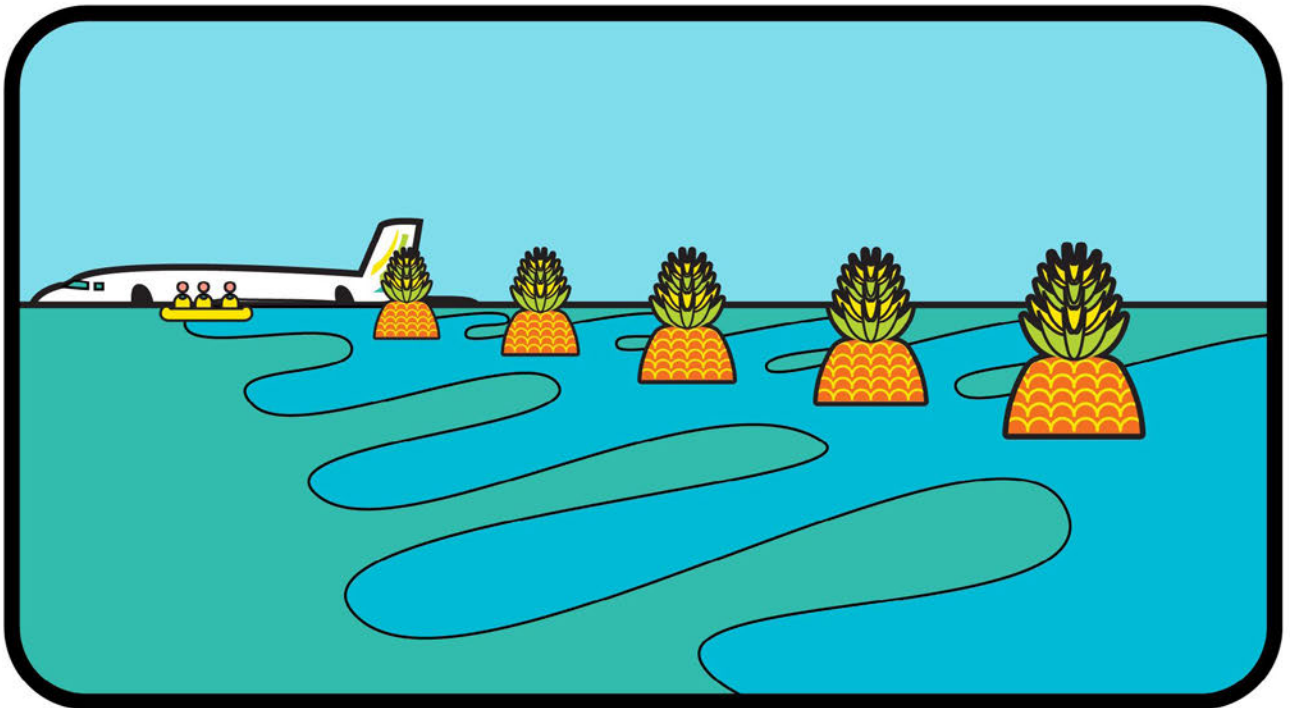
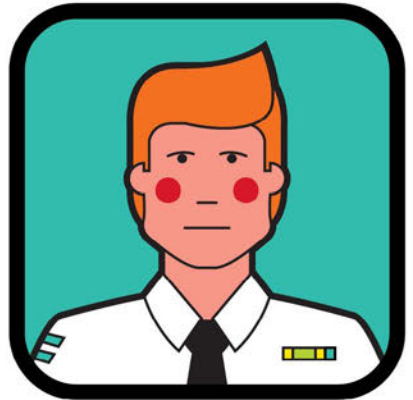
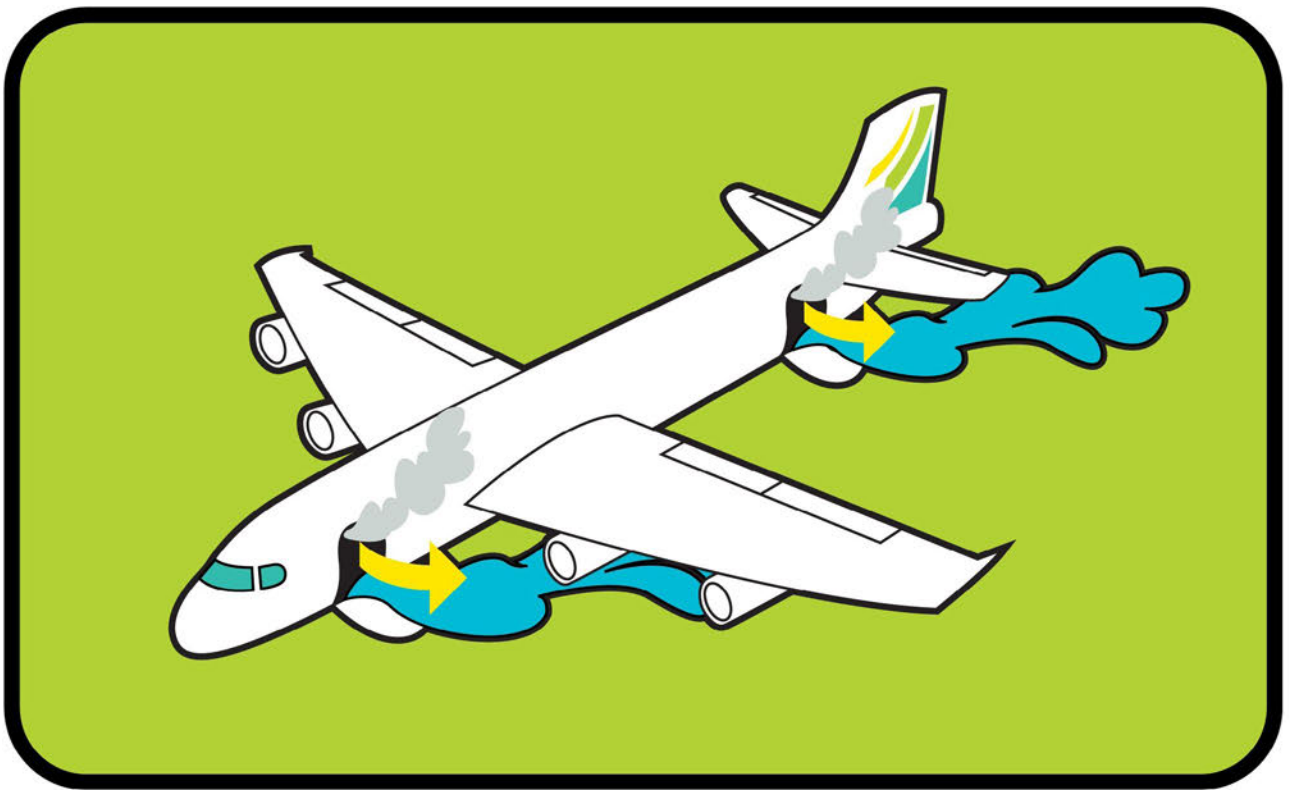
By CHRIS PARNIN

SPECIAL

36 **Sleep: Everything You Need To Know**

By MAROUN NAJJAR





The Exploding Toilet

By PATRICK SMITH

ONE OLD ADAGE defines the business of flying planes as long stretches of boredom punctuated by moments of sheer terror. Moments of sheer ridiculousness, maybe, are equally as harrowing. One young pilot, when he was 22 and trying to impress the pretty Christine Collingworth with a sightseeing circuit in a friend's four-seater, highlighted their date by whacking his forehead into the metal pitot tube jutting from the wing. Earning a famous "Cessna dimple," so he chose to think, would be the stupidest thing he'd ever do in or around an airplane.

That was a long time ago, and a long way from this same pilot's mind during a late-night cargo flight in the winter of 1998:

It's eleven p.m. and the airplane, an old DC-8 freighter loaded with pineapples, is somewhere over the Bermuda Triangle, bound from San Juan, Puerto Rico to Cincinnati. The night is dark and quiet, void of moonlight, conversation, and for that matter worry. The crew of three is tired, and this will be their last leg in a week's rotation that has brought them from New York to Belgium and back again, onward to Mexico, and now the Caribbean.

They are mesmerized by the calming drone of four high-bypass turbofans and the deceptively peaceful noise created by 500 knots of frigid wind hissing past the cockpit windows. Such a setting, when you really think about it, ought to be enough to scare the living shit from any sensible person. We have no business being up there — participants in such an inherently dangerous balance between naïve solitude and instant death, distracted by paperwork and chicken sandwiches while screaming along, higher than Mount Everest and at the speed of sound in a 40 year-old assemblage of machinery. But such philosophizing is for poets, not pilots, and also makes for exceptionally bad karma. No mystical ruminations were in the job description for these three airmen, consummate professionals who long ago sold their souls to the more practical-minded muses of technology and luck.

Patrick Smith, born Patrick R. Santosuosso of Revere, Massachusetts, a fourth-generation descendant of Neapolitan olive growers, is one of these consummate professionals. He is the second officer. His station, a sideways-turned chair and a great, blackboard-sized panel

of instruments, is set against the starboard wall of the cockpit. Now 34, Patrick has seen his career stray oddly from its intended course. His ambitions of flying gleaming new passenger jets to distant ports-of-call have given way to the coarser world of air cargo: to sleepless, back-of-the-clock timetables, the greasy glare of warehouse lights and the roar of forklifts — realities that have aroused a low note of disappointment that rings constantly in the back of his brain.

All is quiet, somewhere around mid-flight, when he stands from his seat and walks out of the cockpit, closing the door behind him. Here he enters the only other zone of the aircraft that is accessible during flight, the small entryway vestibule adjacent to the main cabin door. It contains a life raft, oven, cooler, some storage space and the lavatory. His plan is simple enough — to get himself a Diet Coke. The soft drinks are in a cardboard box on the floor, in a six-pack strapped together with one of those clear plastic harnesses so threatening to sea turtles and small children. These plastic rings are banned at home, but apparently perfectly legal in the Caribbean, where there are, of course, lots of sea turtles and small

“He turns and looks at the toilet. But it has, for all practical purposes, disappeared.”

children. The pilot thinks about this as he reaches for a can, weighing the injustices of the world, philosophizing, daydreaming, ruminating — things that, again, his manuals neither command nor endorse, for perhaps good reason.

He unstraps a Coke and decides to put the remaining ones in the cooler to chill. The cooler, a red, lift-top Coleman that you'd buy in Walmart or Sears, sits in front of the lavatory and is packed with bags of ice. He drops in the cans, but now the cooler will not close. There's too much ice. One of the bags will have to go. So he pulls one out and shuts the lid.

Decisions, decisions. Which checklist do I initiate? Which valve do I command closed? Which circuit breakers do I pull? How do I keep us alive and this contraption intact? And what to do, now, with an extra, sopping wet bag of ice? Well, the pilot will do what he always does with an extra bag of ice. He will open the bag and dump it down the toilet. This he has done so often that the sound of a hundred cubes hitting the metal bowl is a familiar one.

This time, though, for reasons he hasn't realized yet, there are

no cubes. More accurately, there is one huge cube. He rips open the bag, which is greenish and slightly opaque, and out slides a long, single block of ice, probably two pounds' worth, that clatters off the rim and splashes into the bowl. There it is met, of course, by the caustic blue liquid one always finds in airplane toilets — the strange chemical cocktail that so efficiently and brightly neutralizes our organic contributions.

The fluid washes over the ice. He hits the flush button and the block is drawn into the hole and out of sight. He turns, clutching the empty bag and worrying still about the dangers of plastic rings and turtles, picturing some poor endangered hawksbill choking to death. It's just not fair.

And it's now that the noise begins. As he steps away, the pilot hears a deep and powerful burble, which immediately repeats itself and seems to emanate from somewhere in the bowels of the plane. How to describe it? It's similar to the sound your own innards might make if you've eaten an entire pizza or, perhaps swallowed Drano, amplified many times over. The pilot stops and a quick shot

of adrenaline pulses into his veins. What was that? It grows louder. Then there's a rumble, a vibration passes up through his feet, and from behind him comes a loud swishing noise.

He turns and looks at the toilet. But it has, for all practical purposes, disappeared. Where it once rested he now finds what can best be described only as a vision. In place of the commode roars a fluorescent blue waterfall — a huge, heaving cascade of toilet fluid thrust waist-high into the air and splashing into all four corners of the lavatory. Pouring from the top of this volcano, like smoke out of a factory chimney, is a rapidly spreading pall of what looks like steam.

The pilot closes his eyes tightly for a second, then reopens them. He does this not for theatrics, or to create an embellishment for later use in a story. He does it because, for the first time in his life, he truly does not believe what is cast in front of him.

The fountain grows taller, and he sees that the toilet is not spraying so much as it is bubbling — a geyser of lathering blue foam topped with white fog. And suddenly he realizes what's happened. It was not a block

“Like the smoke show at a rock concert, the cabin continues filling with white vapor.”

of ice, exactly, that he fed to the toilet. It was a block of dry ice.

Dry ice is solid carbon dioxide, and to combine it with liquid is to initiate the turbulent and rather unstoppable chemical reaction now underway before our unfortunate friend. The effect, though in our case on a much grander scale, is similar to the mixing of baking soda with vinegar, or dumping water into a Friolator, an exciting experiment those of you who've worked in restaurants have probably experienced: the boiling oil will have nothing to do with the water, discharging its elements in a violent surge of bubbles. Normally when the caterers use dry ice, it's packed apart in smaller, square-shaped bags you can't miss. Today, for whatever reason, an extra-large allotment was stuffed into a regular old ice cube bag — two pounds of CO₂ now mixing quite unhappily with a tankful of acid.

Within seconds a wide blue river begins to flow out of the lavatory and across the floor, where a series of tracks, panels, and gullies promptly splits it into several smaller rivers, each leading away to a different nether-region beneath the main deck of the DC-8. The

liquid moves rapidly along these paths, spilling off into the crevices. It's your worst bathroom nightmare at home or in a hotel — clogging up the shitter at midnight and watching it overflow — except this time it's a Technicolor eruption of poison, dribbling into the seams of an airplane, down into the entrails to freeze itself around cables or short out bundles of vital wiring. And the pilot knows his cataract is not going to stop until either the CO₂ is entirely evaporated or the tank of blue death is entirely drained. Meanwhile, like the smoke show at a rock concert, the cabin continues filling with white vapor.

He decides to get the captain.

Our captain tonight is a boisterous and slightly crazy Scandinavian named Jens. A tall, square-jawed Norwegian with graying, closely cropped curls and an animated air of imperious cocksure, Jens is one of those guys who makes everybody laugh simply by walking into a room, though whether or not he's trying to is never exactly clear. He is sitting in the captain's chair. The sun has set hours ago but he is still wearing Ray-Bans.

“Jens, come here fast. I need your help.” Jens nods to the first

officer and unbuckles his belt. This is an airline captain, a confident four-striper trained and ready for any variety of airborne calamity — engine failures, fires, bombs, wind shear. What will he find back there?

Jens steps into the alcove and is greeted not by any of a thousand different training scenarios, but by a psychedelic fantasy of color and smoke — a wall of white fog and the fuming blue witch's cauldron, the outfall from which now covers the entire floor from the entrance of the cockpit to the enormous nylon safety net that separates the crew from its load of pineapples.

Jens stares. Then he turns to his young second officer and puts a hand on his shoulder, a gesture of both fatherly comfort and surrendering camaraderie, as if to say, “Don't worry son, I'll clean all this up,” or maybe, “Down with the ship we go.” He sighs, nods toward the fizzing, disgorging bowl and says, with a tone of un-ironic pride: “She's got quite a head on her, doesn't she?”

What can they do? And in one of those dreaded realizations pilots are advised to avoid, that insulation between cockpit calm and atmospheric anarchy looks thin indeed.

An extrapolated horror: the riveted aluminum planks bending apart, the wind rushing in, explosive depressurization, death, the first airliner — no, the first vehicle — in history to crash because of an overflowing toilet. Into the sea, where divers and salvage ships will haul up the wreckage, detritus trailing from mauled, unrecognizable pieces while investigators shake their heads. At least, the pilot thinks, odds are nobody will ever know the truth, the cold ocean carrying away the evidence. He's good as dead, but saved, maybe, from immortal embarrassment. A dash of mystique awaits him, the same that met Saint-Exupéry at the bottom of the Mediterranean, another lousy pilot who got philosophical and paid the price. Maybe he blew up the toilet too. Probable cause: unknown.

"Call flight control," commands Jens, hoping a dose of authority will interject some clarity into a scene that is obviously and hopelessly absurd. "Get a patch with maintenance and tell them what happened."

The pilot rushes back to the cockpit to call the company's maintenance staff. He fires up the high-frequency (HF) radios, small black boxes that can bounce the human voice, and any of its associated embarrassments, up off the ionosphere and halfway around the world if need be. Trouble is, he will announce his predicament not only to the mechanics, but also to any of dozens of other crews monitoring the same frequency. Even before keying the mike he can see the looks and hear the wisecracks from the Delta and United pilots in their state-of-the-art 777s, Mozart soothing their passengers through Bose headsets, flight attendants wiping

down the basins while somewhere in the night sky three poor souls in a Cold War relic are trapped in a blue scatological hell, struggling helplessly with a flood of shit and chemicals.

"You say the toilet exploded?" Maintenance is on the line, incredulous but not particularly helpful. "Well, um, not sure. Should be okay. Nothing below the cabin there to worry about. Press on, I guess." Thanks. Click.

Jens has now grabbed the extension wand for the fire extinguisher — a hollow metal pole the length of a harpoon — and is shoving it down into the bowl trying to agitate the mixture to a stop. Several minutes have passed, and a good ten gallons have streamed their way onto the floor and beyond.

Up front, the first officer has no idea what's going on. Looking behind him, his view mostly blocked by the circuit breaker panels and cockpit door, this is what he sees: a haze of white odorless smoke, and his captain yelping with laughter and thrusting at something with a long metal pole.

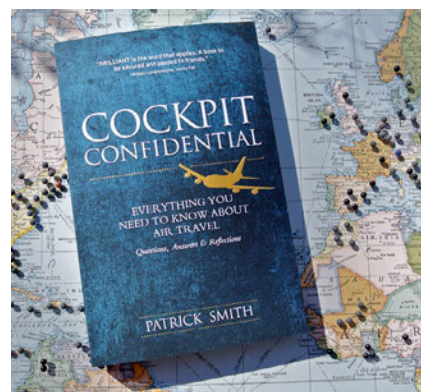
The pilot stands aside, watching Jens do battle. This was the same little kid who dreamed of becoming a 747 captain, the embodiment of all that was, and could still be, glamorous and exciting about aviation. And poor Jens, whose ancestors ploughed this same Atlantic in longboats, ravenous for adventure and conquest. Here he is, a twenty-first century Viking jousting with a broken toilet.

So it goes, and by the time the airplane touches down, its plumbing finally at rest, each and every employee at the cargo hub, clued in by the amused mechanics who received our distress call, already

knows the story of the idiot who poured dry ice into the crapper. His socks and hundred-dollar Rockports have been badly damaged, while the walls, panels and placards aboard aircraft 806 are forever dyed a heavenly azure.

The crew bus pulls up to the stairs, and as the pilots step on board the driver looks up at him. There's a knowing look in the driver's eye. "It was you!" he says excitedly, "Wasn't it?" ■

Patrick Smith is an airline pilot and the host of *askthepilot.com*. He lives near Boston. His new book is "Cockpit Confidential: Everything You Need to Know About Air Travel" [hn.my/cockpit].



Reprinted with permission of the original author.
First appeared in hn.my/askpilot (askthepilot.com)

Illustration by Ben O'Brien.



Now you can hack on DuckDuckGo

DuckDuckHack

Create instant answer plugins for DuckDuckGo

duckduckhack.com



Growing Tomatoes

By JOE HEWITT

WHEN PEOPLE ASKED me what I was going to grow in my new garden, my response was, "Anything but tomatoes."

Blame it on my Italian relatives for excessive use of marinara, or on American restaurants for the pale, mealy wedges atop salads, but I just don't like tomatoes. I was excited about growing peaches, blueberries, broccoli, and just about anything else.



From seed trays to 4-inch pots to 1-gallon pots.

I'm fortunate to live near Love Apple Farms in Santa Cruz, California. Their main gig is growing produce for a Michelin-starred restaurant, but they also teach an array of gardening classes. After having a great time in their spring vegetable class, I returned excitedly to the class roster to find another. Given that the name of the farm, "Love Apple," is an antiquated French name for tomatoes, you can probably guess which plant most of the classes were about. I signed up for one nonetheless, convinced that only a selfish gardener would refuse to grow things they don't enjoy eating.

Seed Party

The farm's owner, Cynthia Sandberg, opened class with a refrain I'd heard before: homegrown heirloom tomatoes are infinitely superior to the supermarket variety. I wanted to believe her, but I wasn't optimistic.

And then she began passing around hundreds of jars of seeds for us to choose from — Black Zebra! Green Giant! Amazon Chocolate! — and my collector's instinct kicked in as I imagined the menagerie of fruit that would burst from Cynthia's catalog into my garden.

Selecting which seeds to plant was kind of stressful. The jars swapped hands so fast, I barely had time to pinch out seeds, much less discern orange striped this from sweet red that. I ended up shooting for a mix of colors and shapes and hoping for the best.

My Tomato Pets

Once I got home, I learned that tomatoes are more like pets than houseplants. The fun of watching them sprout soon gave way to despair as many seedlings wilted and died when I forgot to check them even for a day or two.

Despite having all the recommended equipment — a heating mat, grow lights, a fan, timers, and temperature sensors — they mostly just needed personal attention (and water). As they grew, the seedlings had to be carried into the sun during the day and brought back in at night. Once they outgrew their trays, I had to transplant them to larger pots, and soon after, still larger 1-gallon pots.

I hadn't considered what I was going to do with 100 seedlings until I was at the nursery buying 100 plastic pots. My garden is big, but not that big. Neglect thinned the numbers quite a bit, but I still wound up with more than 60 plants in 1-gallon pots. A dozen or so found their way to friends' gardens, and somehow I found room for 25 plants in the ground.

Fish Heads, Aspirin, Rusty Cages

Twenty-five plants may not seem like a lot until you try to lift a coil of 25 steel tomato cage — each 7 feet tall — from the bed of a pickup. Cynthia insisted on this height and even predicted the vines might grow taller and spill over (indeed, they would). Before I could set up the cages, I needed to get the tomatoes in the ground. Of course, my needy pets insisted I first round up a witch's brew of soil amendments.

According to Cynthia's prescription, each young plant should be buried with the following: one fish head, a handful of egg shells, bone meal, worm castings, tomato fertilizer, mycorrhizal fungi, and two aspirin. (Yes, aspirin. Apparently it combats disease.)



Only thing missing was frog's legs.

After three months of babying the seedlings, I wasn't about to slack off now. It took a few phone calls to track down the fish heads, but Whole Foods came through for me with beautiful 1-pound salmon heads. Into the ground they went, up went the cages, down went the irrigation tubes, and then the wait began.

Nothing seemed to happen for weeks at a time, and I kind of forgot about them for the summer. When the fruit began to ripen in August, there was suddenly a lot happening, and not all of it good. I learned a few things the hard way.

Lesson #1

Cages Are There For A Reason

Tomatoes are vines, and when vines spill out of their cages, they need to be pushed back in. Otherwise, tomatoes will soon be growing in hard-to-reach places, or worse, rotting on the ground. Young and limber vines are easily snaked back into the cage, but mature woody vines will only snap if you try this.

Lesson #2

Treat Blossom-End Rot

About half the plants had fruit with "blossom-end rot," a condition caused by poor calcium absorption. The egg shells were supposed to help with this, but I guess I didn't use enough. There's tons of literature on the web on blossom-end rot prevention — this year I'll be on top of it.

Lesson #3

Fear the rain

Coastal California summers are bone-dry, which is great for tomatoes. But in late August, we had an unexpected downpour. My vines were covered in almost-ripe fruit that I was waiting to get super-ripe, but in the days after the rain many of them began to split and rot. Lesson learned: when it rains, immediately harvest everything that is remotely edible.

Lesson #4

Know Your Diseases

In early September, many of the plants began to look parched, with droopy and desiccated leaves. I thought they just needed more water, but I was mistaken. The dead leaves were from "late blight,"

a dreaded tomato disease. Guess what "late blight" loves? Water! Oops.

Before I realized my error and stopped the irrigation, a couple plants were killed to the ground. Others stopped fruiting, and what fruit was left didn't ripen. Fortunately, only about 1/3 of the plants were affected. This year I'll definitely back off on water later in the season.

Lesson #5

Be Ready For An Avalanche

Growing the plants was hard work, I thought, but that was nothing compared to the work of harvest. My four cherry tomato plants started ripening in early August, and I expected the rest to steadily follow for the next couple of months, giving me plenty of time to deal with them. Instead, my other 21 non-cherry plants ripened all at once in early September, and I had about three weeks to deal with the bulk of them.



A typical day in September.

September was a tomato-chopping, crushing, canning, freezing, and dehydrating frenzy. I ate tomatoes with nearly every meal and delivered heaping baskets to friends and neighbors. With my freezer and pantry overflowing, the tomatoes still kept coming. I was greeted with the smell of mold each morning as I walked past the growing backlog in my kitchen. The first time I had to compost a moldy tomato was sad, but by mid-September, every lost tomato was kind of a relief. The lesson here is that 25 plants are too many for me. I'll probably have eight this year.

But How Do They Taste?

You might be wondering at this point if I changed my mind about the merits of eating “love apples.” I'll say this much: a tomato, picked when warm from the sun, so ripe it's on the verge of going bad, and eaten while standing in the garden, is better than ice cream. On the other hand, some of the varieties I grew, despite being fancy heirlooms, tasted no better than a winter greenhouse tomato from Safeway.

I conducted a taste test with my parents with each of the 25 varieties. One clear winner emerged, and that is Orange Russian 117, which is not only the most beautiful tomato I've ever seen, but tastes more like a peach than a tomato. Other favorites included Coyote and Sungold cherries; the hefty Casey's Pure Yellow and German Red Strawberry; the striped Tigerella; and George O'Brien for making sauce. Some, like Black Plum, tasted just OK when raw, but were spectacular when dehydrated.

Until next year

As it turns out, I was right all along. Tomatoes aren't that great — unless you grow the right varieties, eat them at the right time, and process them the right way. As I ripped out the last of the tomatoes in October, it was clear I was ending a cycle that would begin again every year for as long as I'm fit to garden. ■

Joe is a programmer and novice gardener. He's worked on Firefox, Firebug, and Facebook's iPhone app.

Reprinted with permission of the original author.
First appeared in hn.my/tomato (joehe Witt.com)

Illustration by Parko Polo.

Making A Physical Product

By JON WHEATLEY

I HAD BEEN KICKING around the idea for some kind of space-themed dice game for a while. I thought it would be a really nice metaphor for what actually happens when galaxies are formed. The dice represent balls of matter floating around the universe. Sometimes they bump into other balls of matter and become stuff.

I decided to design the game around the “press your luck” genre. The concept of these games is usually pretty simple. Some things are good to roll and some things are bad. The goal is to roll as many good things as you can before rolling a bad thing and resetting your score. The first person to get X good things is the winner.

The goal with Space Dice is to roll as many “habitable planets” as possible, before you roll 3 black holes and make your galaxy unstable. A “habitable planet” is a planet near at least 1 star and without space debris hurtling towards it. The first person to roll 10 habitable planets is the winner.

Here’s a VERY early prototype (figure 1) of the game I was working on over Christmas. It really

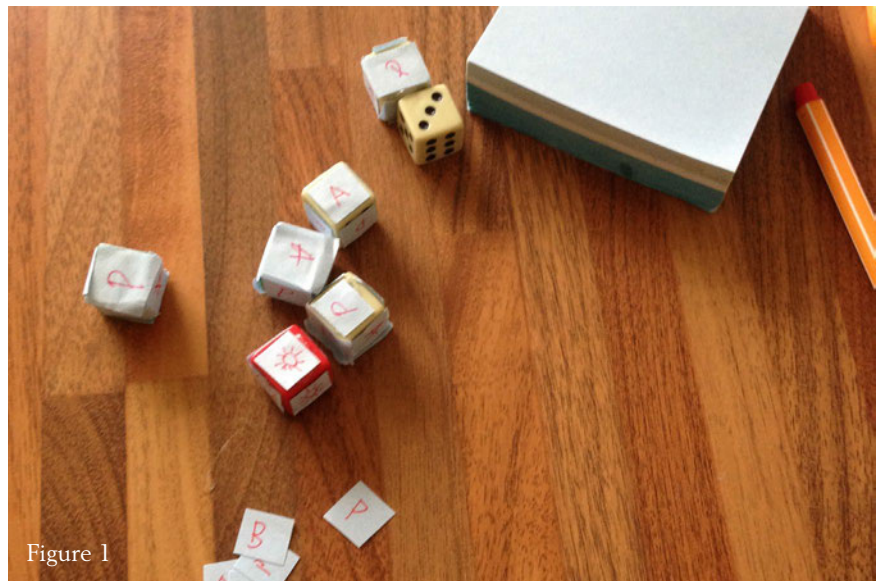


Figure 1

helped to make a prototype, as I could actually play the game rather than everything being hypothetical.

Having finalized the basic concept, I needed to make sure the game would actually be playable. There should be a good chance of rolling the things you need to roll to score. It shouldn’t be likely to die in one roll. Rolling a supernova (which destroys every planet every player has scored up until that point) shouldn’t happen very often, etc.

I spent a few hours hacking together a little rails app which

would play out 100,000 rolls in a few seconds and tell me the chances of everything coming up. The script let me easily play around with the values on each of the dice and tweak the chances of everything happening. It’s still online if you want to look at it [spacedice.herokuapp.com] and the code is on GitHub [hn.my/spacedice]. Please excuse my bad Ruby code.

I hired a graphic designer to make some nice icons. I was ready to start talking to factories.



Figure 2

I looked on *alibaba.com* and found some dice manufactures in China. It turns out 12 sided dice are MUCH more expensive than regular 6 sided dice (about \$0.25 per dice vs. \$0.05). I reluctantly decided to go with the 12 sided dice anyway, because the shape of them worked much better with the theme. Rarely do you see celestial bodies that are square!

The factory did a test print and made a complete set of space dice. (figure 2) Now it was starting to become real!

The factory was happy to ship out the tester dice they made before entering full production. Luckily, this coincided nicely with Chinese New Year so I had a bit of time to play test the game and request any changes. When I received the package, I excitedly opened it and anxiously waited for my girlfriend to get home so we could play Space Dice for the first time.

I may be a little biased here, but the game was great fun. Everything worked as intended, and my girlfriend picked it up pretty quickly. The only thing that was a little awkward was keeping score of how

many habitable planets each of us owned. We solved this by using a pen and paper to keep track, but I'm planning on building a simple Space Dice score keeping iPhone app at some point.

The next step was the packaging. I found a factory in LA [spiralpaper.com] that made custom tubes with removable tops. This was good because the tube could also double as the dice shaker. They were nice enough to send over some pictures of the tubes being made. (figure 3)

The last thing I needed was instructions. This turned out to be a much harder task then I was anticipating. There were a lot of things that seemed obvious to me that absolutely were not obvious to other people when I spoke to them. It was also hard to explain all the rules, including rare edge cases, while keeping the instructions short and not intimidating to new players. I needed professional help.



Figure 3

I hired a freelance copywriter to give me a hand, which made things much easier. There was still a fair amount of back and forth until we decided on the best way to structure and word everything but we eventually hammered out some pretty decent instructions.

They just needed some design and they were ready.

Everything arrived at my apartment, and I started packaging sets together. This took much longer than I thought it would (1.5ish minutes per set, 333 sets, just over 8 hours).

And here it is. The final, completed set of space dice in all its glory.



Shipping

As one might expect with something like this, shipping is a huge time sink. The first few days after launching were spent almost entirely packaging space dice sets and hand writing addresses. It wasn't long before our daily trips to the post office were met with audible groans from the staff there. Each package needed to be individually weighed and stamped. For international orders the addresses on the customs forms needed to be manually typed in. It was horrible.

I looked into a few different options. I really wanted to automate the whole process so I looked into local fulfillment services. Unfortunately at such small scale and with the margins I'm currently working with, the math doesn't work. It would cost a few dollars extra per order (I received quotes ranging from \$3-10) on top of a monthly storage fee.

I now have a dedicated shipping computer (~\$300 from OfficeMax). My friend recommended getting a zebra label printer which uses thermal paper to print labels rather than ink (~\$100 from Amazon) and a free USB scale from stamps.com.

Now shipping takes a fraction of the time. Stamps can be printed rather than arduously hand written and customs forms are automatically filled in. The shipping software also spits out a free tracking number which can be passed onto the customer. All that and USPS will come and collect!

Reception

The game was mostly well received although a few weeks in I started to get a few reports that there was a bug.

In the game it's possible to roll a "supernova" which completely resets everyone's scores. In my simulations and play testing this rarely happened (there's about a 6% chance). I enjoyed the rule because it meant however far behind you got there was always an outside chance you could roll a supernova and reset the scores so you're level with your opponent again. Unfortunately in all the play testing I did I only ever played the game with one other person.

This was a big mistake.

The problem is the more people you play the game with the more likely it is to roll a supernova during

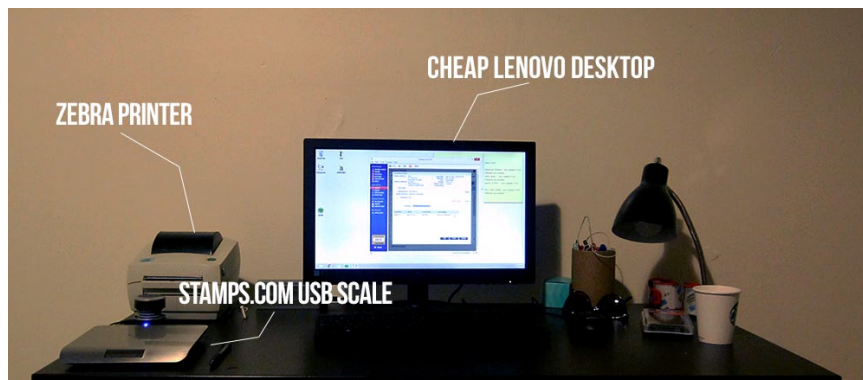
a round. There's a point where it becomes more likely than not that a supernova will be rolled essentially making the game last forever. This was a huge oversight and something I'm sure any professional game designer would have noticed instantly.

Luckily it's a relatively easy fix (the rules are being tweaked so a supernova just resets your score and not everyone's) but this was still a harsh lesson in the value of play testing more than you think you should.

Despite the bug, lots of people were enjoying the game. A few people sent me pictures of themselves or their families playing which was great to see. Someone even made a really cool score card app [spacedicescorecard.com] which lets players easily track scores in the game.

I had some success approaching local toy stores and asking if they'd like to carry the game. Out of 80 stores contacted so far, 10 were interested in doing a small test run of the game to see how it performed in the store. I didn't do anything fancy at all here. I just searched Yelp for toy stores, found their website, then their email, then cold-emailed them and told them some high level info about the game. Considering my somewhat unsophisticated approach, I think a 12.5% potential conversion is pretty good. ■

Jon Wheatley is a British born, San Francisco based entrepreneur. Previously founding DailyBooth, and currently working on new projects. Head over to spacedice.com if you'd like to buy a copy of Space Dice.



Reprinted with permission of the original author.
First appeared in hn.my/physical (jonw.com)

stripe

Accept payments online.

The Freelancer's Guide To Recurring Revenue

By BRENNAN DUNN

IT'S PRETTY MUCH inevitable that after the realization that "Oh my God, I'm making a ton of money" freelancing, just about every consultant I've met soon realizes, "But that money disappears when I stop working."

This article is going to dive into how you, an idea-less freelance consultant, can build products of your own and develop recurring revenue streams that don't necessarily require an ongoing commitment of your time.

Why Recurring Revenue?

Recurring revenue is much more than just "money that will come to me as I sip Mai Tais on Waikiki Beach." Recurring revenue is (more or less) predictable revenue. The catch with consulting is that the majority of us bill for our time, and if no time is logged — either because we're on vacation or simply just don't have a project to work on — there's nothing to bill for. But we still have monthly obligations. We still need to pay for our shelter, our food, and other amenities required to live comfortably.

I can't speak for you, but if I'm on the hook for paying banks, creditors, or utility companies upwards of \$10,000 for the foreseeable future, being uncertain about where I'm going to get that \$10,000 is a huge weight to shoulder. Our salaried brethren understand what it feels like to be worried about getting fired and losing their paycheck; but many of us freelancers constantly face that fear, and that's a fate that can occur despite how successful we are at the work we produce for our clients.

What attracted me to pursuing recurring revenue was that I could offset my fixed liabilities with something that I had a lot of control over. If one of my two current consulting clients bailed, my income might be cut in half by 50%. If one customer out of the thousands that have bought a book of mine ask for a refund, I might need to skip dessert tonight.

Products: An Emotional Hurdle To Overcome

I was in Las Vegas last week for MicroConf, an annual conference for bootstrapped entrepreneurs. And I noticed a trend: everyone there was aiming to one day live entirely off of product income, but a significant majority was using consulting as a crutch to support them until they hit some critical mass of paying customers that they could cut the umbilical cord of consulting (keep in mind this is a bootstrappers conference — we don't have the luxury of the massive amounts of cash infusion that can come through funding!)

The jump, however, between selling time for cash and selling a self-serve product to someone you've never met is pretty big.

The former just requires you finding a few people like you who can effectively "rent" you for a rate, either because they don't have whatever skill they hired you for or they don't have the time to do it themselves.

The latter can require figuring out a problem that people will pay YOU, an Internet stranger, to fix. This means marketing, writing persuasive sales copy, and building a product before anyone's bought it (which can be a struggle for those of us conditioned to "work now, get paid soon-ish") and having the gumption to actually ask for the sale.

Marketing. Persuasion. Building a product. Fulfillment. These dragons are slain through research and elbow grease. But before any of this can happen, many freelancers-turned-product-people need to come up with an idea, which is where most of us stumble and give up.

The problem is the belief that ideas are special, and that "great ideas" will come given enough time (or enough showers, depending on when and where you get inspired.)

But what if you didn't have to wait. What if you already had the great ideas you need to build up recurring revenue?

Sell What You Know

I want to talk about two different audiences you can sell to right now, and how you can go from trading your time for money to building up a product that literally sells itself while you sleep. As an added bonus, doing this can help you get more clients, which helps alleviate the "feast or famine" problem that affects a lot of freelancers.

To begin, you need to start with a little soul searching...

Why do clients pay you wads of cash? The answer isn't necessarily "Because I write code" or "I'm good at setting up WordPress." Most people will hire you because you're capable of solving some sort of business problem that makes it

worthwhile to cut you large checks with the assumption that you'll deliver something worth more than the cost to hire you.

So if they're hiring you to, say, get more walk-in customers to their retail shop by setting up some really solid stuff online, that's at the far end of the execution spectrum ("Here's my problem, you're the tech guy, I'll pay you to fix it.") You're consulting. You're applying your knowledge to solve one person's pain, which in this case is a lack of customers which is causing a bit of a problem with their cash flow.

Alternatively, the skills you possess that allow you to charge your clients to get them more walk-in customers might be desired by a more junior freelancer who also wants to become more valuable by offering this service to their clients.

How did you get to where you are today? What skills did you need to acquire before you were able to make yourself valuable enough to be hired? And before worrying that you aren't necessarily an expert and getting the maximum amount of walk-in customers for the maximum amount of retail shops, you just need to realize that you only need to know more than the person you're trying to teach.

The theme here is education. Consulting is just applied education, and there's a lot of middle ground between where a client or fellow freelancer is today ("I need more walk-ins" / "I also want to help my clients get more walk-ins") and having an expert consultant apply years of accrued wisdom to a specific problem, or becoming the go-to consultant for getting retail shops more clients.

Sell To Your Peers

Selling information to other freelancers like you can solicit two types of very vocal responses:

You're selling tools to the miners. (The idea being that the only people who made money during the gold rush in the American West were the merchants hawking mining gear to upstart miners. The difference being that in this gold rush, the miners are actually making lots of money.)

...or...

All of this is available for free on the Internet (in existing blog posts, podcasts, forums, StackOverflow, etc.)

But for each of these minority voices there are dozens of silent doers who want what you have to teach so they can move on with their lives, becoming more valuable and charging more.

And these doers will buy the knowledge you've cataloged, organized, and presented for sale. Because when the doer isn't doing, they're not making any money. And if they're billing \$XXX an hour, spending \$XX to level-up is a no brainer. They don't have the time to hunt and peck around Google results to get to the same result that you're offering with a shiny "Buy Now" button.

A great example of this in action is what Nathan Barry's done with his two books, *The App Design Handbook* and *Designing Web Applications*. He's produced two products — the former for iOS designers, the latter for designers of web apps — that can help the reader go from just a designer to being able to walk and talk like an experienced iOS or web app designer.

“Businesses pay for things, especially when those things can reduce costs or amplify revenue.”

The best part of what Nathan's done is that his products are sold and packaged as an easy-to-digest pill. Rather than wading through the good and bad of pages of Google results to figure out how to implement the perfect navigation UI, Nathan's done that work for you and has it available for sale.

Let's think like a CEO who hires designers for a second. Jim, an employee of yours, is a good web designer, but now he needs to design our company's new iPhone app. Sure, he knows how to use Photoshop and splice and dice up PSD files...but Jim's never designed an iOS app. We could think, “Jim, go and figure out the ins and outs of iOS design for a few weeks.” But Jim costs a few thousand dollars a week, and the idea of spending thousands so Jim can (hopefully) become a competent iOS designer sort of sucks. And you don't want to hire that high price consultant and risk pissing off your in-house designer, right?

And then you discover that for \$169, this guy named Nathan has a complete package of resources that help designers become awesome iOS designers. Businesses pay for things, especially when those things can reduce costs or amplify revenue.

Sell To Your Clients

Remember that spectrum, where at one end we have raw information (“The know-how to help a retail business get more walk-ins”) and at the other application (“The knowledge and the time spent to help one retail business get more walk-ins”)?

As freelancers, we're used to being paid to apply knowledge for our clients. But we don't always think about the value we could deliver on the lesser parts of the spectrum.

Think about it this way. To go back to our example, Mary the Business Owner wants more walk-in customers. There are actually quite a few ways to do this:

Learn and act. Mary could read a book or take a training course from someone like you who's constructed a guide for setting up ads and a marketing site that helps convert visitors to walk-in customers. This requires Mary (or someone on staff) to take the time to do some learning and then actually execute on this new knowledge. The product Mary purchases is probably purely information: an eBook, a training course, videos....

Get something turnkey. Mary could come across a SaaS vendor who promises a turnkey website and set of marketing campaigns that are specifically built for retail shops wanting more walk-in customers. This is significantly lower risk (and cost) for Mary, because the next option would be to hire a designer off the street and hope that he 1) knows how to build the site and the marketing campaigns, and 2) knows enough about retail shops and how they get walk-in customers. Chances are, most of the products you've wanted to build have fallen into this space. The product Mary purchases is probably a web-based, multi-tenant product that she pays for monthly.

From scratch. Mary needs her problem solved, and she lacks the technical capacity or time and energy to do it herself. So she finds someone like you, is persuaded that you have the skills to solve her problem, and then commissions you after a high-touch, possibly lengthy sales cycle. The product Mary purchases is a one-off consulting engagement with someone (an individual or agency) that she feels can help solve her problem.

An added benefit of educating your client is that you'll increase your perceived expertise, which could end up netting you new clients who need to hire a consultant who helps retail businesses get more walk-in customers. After all, you literally wrote the book on it.

Patrick McKenzie is routinely hired by software companies to help improve their conversion rates through something called lifecycle emails. Now, Patrick lives in Japan, and just about all of his clients are located halfway around the globe from him. Hiring Patrick involves going through a proposal, possibly convincing partners or a board to hire this guy for \$XX,XXX a week, accepting his proposal, negotiating contracts, getting Patrick to fly halfway around the world, getting him in your office for at least a week, and then sending him back.

All because you need Patrick to help you apply the email knowledge he has to your business.

This also limits Patrick to having just a handful of clients. He can do the grueling, cross-Pacific flight only so often, and being a newlywed tends to exponentially complicate things.

So while Patrick could just stop consulting, he could also do it at scale. He could distill his knowledge into a five hour video course, and allow people that couldn't afford him or win a coveted spot in his consulting schedule to reap a lot of the benefits of having Patrick sit in your office for a week.

The Middle Ground Between Information and Application

Where Patrick hasn't gone (and where, frankly, I think he and you could) is the middle ground between teaching and doing.

When Patrick is hired to write lifecycle emails for a software company, the hiring company is educated by Patrick, but he's also going to write and put into place the application of that education (e.g., a 30-day email course.) But what happens when a company picks up his video course and takes a stab at implementing the advice contained within?

Who do you think they're wanting to get feedback from?

Patrick. Or if it's an eBook on getting more retail walk-in customers, from you.

And this gives you and Patrick the distinct advantage of being able to charge what might seem an exorbitant rate to spot-check or vet the end result of your customer attempting to take action on your information. And let's face it — only one person really fits the bill to do this one-off consult...the creator of the content that gave them the skills!

Other middle grounds involve group training or workshops, monthly retainer agreements, coaching, and other services that you can price, rather than bill.

To round out the examples, Joanna Wiebe of Copyhackers offers a No-Fluff Website Review for close to \$1,000. The review includes an assessment of your website's copy and a 60-minute video review. Because Joanna's established her foothold in teaching people who hate writing sales copy what it takes to write effective sales copy, her 1-on-1 review is a natural upsell to her books. She'd be hard pressed to sell this service of hers without the book, but again — once someone's spent a few hours in Joanna's head by reading her book, she's going to be the only viable candidate that's capable of conducting a website review.

So before you get dismayed by the amount of time and effort it takes to build a product, realize that products can be as simple as something that teaches just one thing. And instead of waiting for your Eureka moment, reflect on why people have hired you in the past. The road to selling products doesn't need to be steep and jagged. ■

Brennan and Patrick McKenzie will be running a "Recurring Revenue (for freelancers and consultants)" workshop this month. For more, see recurringrevenueforconsultants.com

Brennan Dunn is an author, teacher, startup founder, podcast host, and newsletter junkie. He's written two books, "Double Your Freelancing Rate" and "Sell Yourself Online: The Blueprint". Brennan is also the founder of Planscope, a project management tool for freelancers and small teams.

Reprinted with permission of the original author.
First appeared in hn.my/recurring (planscope.io)

Vim After 11 Years

By IAN LANGWORTH

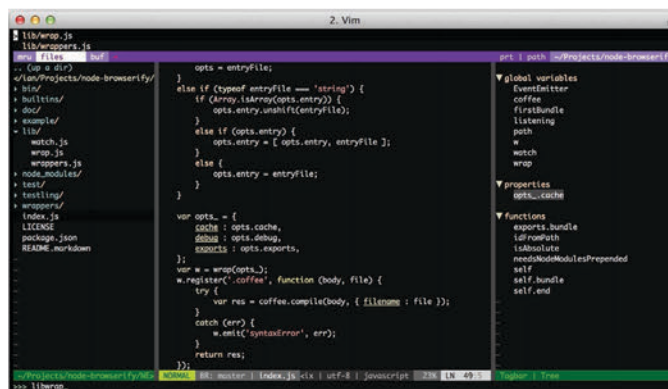
AT SOME POINT over a decade ago I received my first real Unix account on Northeastern CCS's computing infrastructure. I realized that my primary method of development — editing files in BBEdit and uploading them via FTP — wouldn't scale for college-level projects, so I decided to learn how to efficiently edit files on a remote host. I used Pico for a while but became annoyed at its lack of syntax highlighting, so I used the only other editor I remembered bumping into: Vim.

Fast forward eleven years. I've written many different languages and used Eclipse, IntelliJ IDEA, TextMate, and Flash Builder. I even used Emacs for a year just to see what I was missing. But when it was time to start a company, I knew that an early priority would be to bang out code quickly, so I chose the best tool for the job.

Recently, someone noticed a plugin I was using and said, "I had no idea Vim could do that. You should teach me all of these tricks someday." A fine suggestion, indeed, and that day is today.

Customizing Vim: A Preamble

Customizing Vim is two parts: editing rc files and installing plugins. You should already have a `.vim/` directory with a `vimrc` and possibly a `gvimrc`. For plugins, the first thing to look at is Pathogen



[hn.my/pathogen], a neat little plugin that lets you keep plugins in subdirectories of `.vim/` instead of merging everything into the big `.vim/` tree. It's easy to install and makes it a breeze to try out new plugins.

It's also useful to keep the `.vim/` directory in

version control because you can test a plugin or setting and, if you don't like it, put everything back the way it used to be with a `git revert`.

Additionally, I use an `update.sh` script [hn.my/updatesh] I wrote to install plugins as well as keep everything up to date. Keeping Vim plugins up to date is advantageous; I've found that Vim plugins are often updated to fix bugs and not change functionality significantly.

Regarding mappings, I use the backslash key as my utility prefix since backslash isn't bound to anything by default. For example, `\o` toggles paste mode and `\l` toggles line numbers:

```
:nmap \l :setlocal number!<CR>
:nmap \o :set paste!<CR>
```

Rudimentary Essentials

Vim's defaults are pretty smart, but there are a few small rough spots which need ironing out.

The `Esc` key is used to return to Normal mode, but on most keyboards the `Esc` key is pretty far from the

home row. `Ctrl-\\` produces the same keycode but involves two hands. Both might have problems on latent terminals which support `Esc` as an alternative to `Meta`. Alternatively, use `Ctrl-C` — it's easy to type, instant, and you're already used to jamming on it from when you cancel commands. The only caveat is that you still need to use `Esc` for operations in visual block mode (more on this later).

Moving around text is part of what makes using Vim feel so efficient. The one single thing I could never stand about Vim's default movements, however, is how `j` and `k` move around wrapped lines. By default they move you one line down and up but on a linewise basis, which is annoying. If I hit `j` I would expect the cursor to move down a single row on the screen, just like every other text editing area in the world. The following does just that:

```
:nmap j gj
:nmap k gk
```

Regarding Vim's command line, its defaults make it behave very unlike a modern command line. If you're used to Emacs-style movement keys on your Bash or Zsh command line (using `Ctrl-A` and `Ctrl-E` and the like) then you might want to make Vim act the same way:

```
:cnoremap <C-a> <Home>
:cnoremap <C-b> <Left>
:cnoremap <C-f> <Right>
:cnoremap <C-d> <Delete>
:cnoremap <M-b> <S-Left>
:cnoremap <M-f> <S-Right>
:cnoremap <M-d> <S-right><Delete>
:cnoremap <Esc>b <S-Left>
:cnoremap <Esc>f <S-Right>
:cnoremap <Esc>d <S-right><Delete>
:cnoremap <C-g> <C-c>
```

A common operation is to search for text, so it makes sense to have some sane defaults. The `incsearch` option highlights as you type an expression (a.k.a. "Emacs style"), and `ignorecase` plus `smartcase` make searches case-insensitive except when you include upper-case characters (so `/foo` matches `F00` and `fOo`, but `/F00` only matches the former). `hlsearch` is a useful option which highlights the current search, but the highlight can become annoying so it makes sense to have a key to clear the highlight when you no longer need it:

```
:set incsearch
:set ignorecase
:set smartcase
:set hlsearch
:nmap \q :nohlsearch<CR>
```

Vim, the Terminal, Buffers, and You.

There are two forms of Vim: Vim in the console and the native gVim application. The advantages of gVim are better OS integration for dialogs, native printing support, and a wider range of color themes. A modern terminal gives you everything else such as mouse support and mostly-decent color.

It was decided early in its development that Vim isn't designed to replace a terminal, so Vim is and forever will be bad at emulating terminals. Using the `:shell` command in gVim is a path to madness. One plugin, Conque, comes close to adding terminal support, but I found that it caused Vim to hang, and its integration felt clunky. Thus, if you do a lot of work from the command line while you edit, the best way to run Vim is from the command line.

One of Vim's strengths is that it starts lightning fast, so starting Vim from the terminal is trivial. With a modern, 256-color terminal like iTerm2 or Gnome Terminal, it will even look like gVim. But the best part is that you can drop into the command line at any time with `Ctrl-Z`, which suspends Vim, and your working directory is where you left off.

If you have trouble remembering which terminal has Vim suspended, try adding the number of background jobs to your shell prompt. If you have many Vims in the background, however, you should start using buffers instead.

Vim has the powerful ability to keep multiple files open in the background, and there are many ways to navigate between them. This is useful for performing work on a project or for editing multiple files as part of an operation.

For example, say your current working directory is a Django project and you want to edit the shopping cart request handler, so you run `vim cart/views.py`. Once Vim is open you realize you need to change a setting so you type `:e settings.py`, which opens up the file `settings.py` in the current window. You can get back to the other file by running `:b views` (`:b` is short for `:buffer`), which performs a substring match across all buffers, or use `:b#`, which opens the previously viewed buffer. Buffers become more powerful once you start using `Ctrl-P`, which I'll get to later.

A lot of times you'll want to do some work somewhere other than the file you're editing and return afterward, in which case `:b#` is a godsend. (Or `:e#`, I'm not sure if there's a difference.) It's so useful that it deserves a more natural keybinding, like `Ctrl-E`. (The default binding is `Ctrl-Shift-6`, which you'll never remember, and nobody knows what `Ctrl-E` does anyway.)

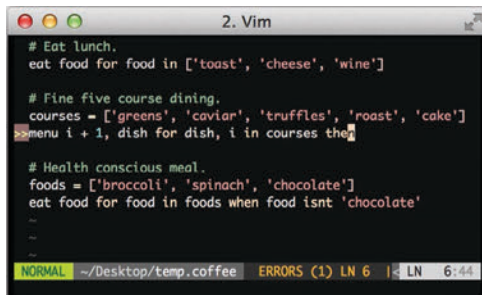
```
nmap <C-e> :e#<CR>
```

Vim automatically creates a buffer for each file on the command line. This is useful from the command line, such as with `vim *.js`, or combined with `grep`/`ack`: `vim `grep -l foo *.js``. I use this pattern so often that I bound two keys to cycle between all open buffers:

```
nmap <C-n> :bnext<CR>
nmap <C-p> :bprev<CR>
```

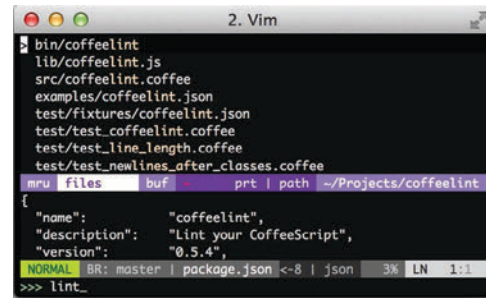
Yes, My Editor Does That

Vim has error highlighting through the plugin Syntastic, [hn.my/syntastic] which uses external compilers and linters to show errors inline with your code. This is absolutely essential, and it works so well that the shortness of my description doesn't do it justice.



Syntastic uses the current file's filetype and runs an appropriate linter or compiler after every save — super useful for finicky syntaxes of JSON, CSS, and SASS, not to mention C.

TextMate raised the bar when it introduced its “Go To File...” command which lets you quickly jump to any file using a fuzzy text search. The best Vim equivalent is `Ctrl-P`, which not only has a fuzzy file search, but a fuzzy buffer search as well:



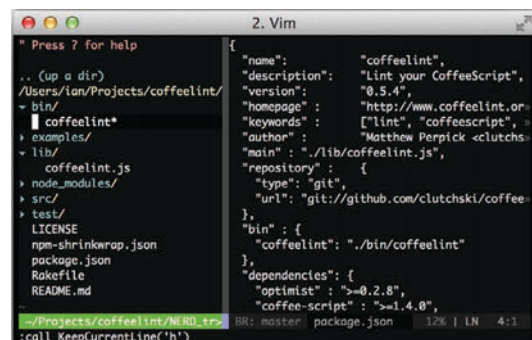
If you use Vim buffers, `Ctrl-P`'s ability to quickly go to the buffer you want is life-changing. It's so useful that I've bound it to `;` (and nobody remembers what `;` does anyway). `Ctrl-P`'s file search combined with buffer search is magnificent; use the file search to open files related to the task at hand, then use buffer search to flip in between them.

```
nmap ; :CtrlPBuffer<CR>
```

It's worth showing a few settings for `Ctrl-P`. The following are the settings I use which map it to `;`, put it at the top of the screen, hide unnecessary files from the search results, and a few more things. Run `:help ctrlp-options` to read more about them.

```
let g:ctrlp_map = '<Leader>t'
let g:ctrlp_match_window_bottom = 0
let g:ctrlp_match_window_reversed = 0
let g:ctrlp_custom_ignore = '\v~$|\. (o|swp|p
yc|wav|mp3|ogg|blend)$|(^|[/\])\.(hg|git|bzd)
($|[/\])|__init__\.py'
let g:ctrlp_working_path_mode = 0
let g:ctrlp_dotfiles = 0
let g:ctrlp_switch_buffer = 0
```

A collapsible directory tree is a great tool to explore a project structure when you don't know what you're looking for or to help keep the project's organization in your head. The best file browser plugin is NERD Tree, [hn.my/nerdtree] which has easy-to-remember keyboard navigation (hit `?` in the window for help), mouse support, and uses little Unicode arrows next to folders.



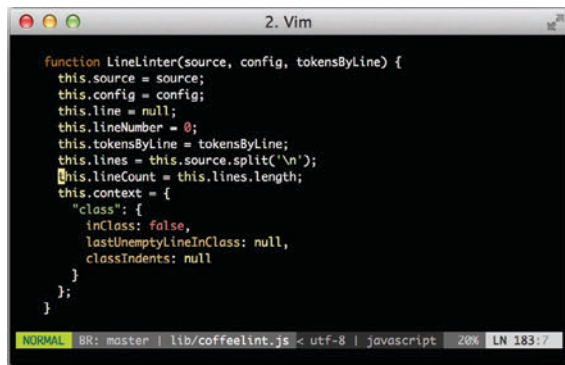
It's also useful to have a key which toggles the visibility of the tree:

```
:nmap \e :NERDTreeToggle<CR>
```

Finally, everybody has a color scheme they're comfortable with. Modern terminals support 256 colors, but sometimes you need to kick Vim to recognize this:

```
if $TERM == "xterm-256color" || $TERM == "screen-256color" || $COLORTERM == "gnome-terminal"
  set t_Co=256
endif
```

You can't, unfortunately, start using any gVim theme immediately, though there's a plugin that tries to make that happen. There exists a plugin called vim-colorschemes [hn.my/vcs] which has bundled hundreds of themes with 256-color support, such as twilight256 and wombat256. It also includes the popular Zenburn theme, but its copy isn't as good as the independently-maintained zenburn.vim [hn.my/zenburn]. If you want to see what all of the themes look like, check out vim-colorscheme page [hn.my/vcspreview] which contains samples of each theme.



Finally, in terms of visuals, Vim's default status line is pretty lacking. A popular plugin is Powerline, [hn.my/powerline] which displays lots of helpful things in the status bar including your current git branch. It also uses colors to cue you into the current mode as well as when paste is enabled.

(Note: As of this writing it seems that the authors of Powerline decided to port the plugin to Python, which unfortunately requires a Vim build with Python support and complicates its installation. The pure vimscript version is still available. [hn.my/powerline])

Other People's Code

If everyone could write code like you, wouldn't the world be a great place? Unfortunately, there are still some jerks who use or don't use tabs, or maybe they indent with four spaces instead of two, or vice versa, or whatever it is that those jerks do. You still need to read their code and it helps to be able to quickly switch between popular (and unpopular) tab modes:

```
:nmap \t :set expandtab tabstop=4 shiftwidth=4
softtabstop=4<CR>
:nmap \T :set expandtab tabstop=8 shiftwidth=8
softtabstop=4<CR>
:nmap \M :set noexpandtab tabstop=8 softtab-
stop=4 shiftwidth=4<CR>
:nmap \m :set expandtab tabstop=2 shiftwidth=2
softtabstop=2<CR>
```

Those authors might wrap or not wrap lines at 80 or 100 columns or whatever it is you like, so being able to quickly toggle wrap mode helps:

```
:nmap \w :setlocal wrap!<CR>:setlocal wrap?<CR>
```

Conclusion

There are still features I miss from Emacs and other editors. For example, Emacs has a useful mode [hn.my/ecss] which highlights hexadecimal colors in CSS and SASS with the color represented by the text. It's smart enough to display light text when the color is dark and vice-versa.

The biggest hole, however, is the lack of refactoring and smart completion. The downside of working with dynamic languages such as JavaScript make writing refactoring engines hard, but even non-language-specific editors like Sublime Text make it easy to rename a variable everywhere within a function. For Python, there's a thing called ropevim [hn.my/ropevim] which adds some refactoring commands, but I've found it to be clunky and unreliable.

Nonetheless, Vim is a great tool, and I hope the above has been useful. ■

Ian Langworth is the co-founder and CTO of Artillery, which aims to bring console-quality games to the browser using HTML5, WebGL and other cutting-edge browser technology. Prior to Artillery, Ian was the first engineering hire at Redbeacon (acquired by The Home Depot in 2012), and before that he was a Software Engineer at Google.

Reprinted with permission of the original author.
First appeared in hn.my/vim11 (statico.github.io)

Yahoo! Chat — A Eulogy

By RIDICULOUS_FISH

“**A**SSWIPE,” REPLIED
YAHOO’S server.
That’s when I knew
I had it.

Yahoo’s public chat rooms have passed away. It is for the best, for the spam had spread everywhere. But they had a good run, operating for a decade and a half, an Internet eternity.

Here are three funny stories from the Yahoo chat protocol.

Body and Zoul

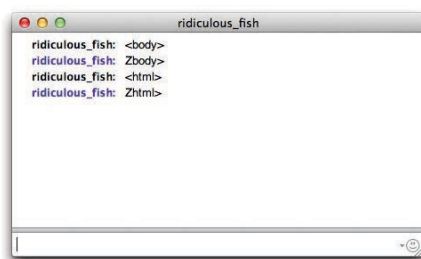
Yahoo chat rooms started life as a Java applet, chugging along in Netscape Navigator 4. Support for chat was later added to Pager, their native app, which did its own chugging in Visual Basic. Initially, Pager had custom text rendering, but then they replaced it with an HTML view.

Oops. Pager didn’t escape message contents, and so it was possible for a message sender to coax the recipient into displaying arbitrary HTML - even fetching images off the web. XSS in its infancy.

Oh dear, what to do? Not everyone would install a security update. But all messages went through Yahoo’s servers, so they could fix

it server-side: block the attack by rewriting the messages before sending them along. So Yahoo replaced the troublesome opening bracket < with a letter that sort of looked like a bracket: a capital Z. Messages containing <html> or <body> would be rewritten to Zhtml> and Zbody>.

And more than a decade later, this methuselan workaround lives on:



md5555555555...

Yahoo chat was not as full of sexually unfulfilled college girls as the spam bots would have you believe. Before the captchas arrived in 2007 (which did little in any case), Yahoo battled the bots by obfuscating the login protocol. And once the bots caught up, obfuscating it again. Rinse and repeat. By the end, the protocol had grown to outrageous complexity. A puny excerpt of the login sequence:

1. md5 the user’s password
2. md5 the password, followed by the fixed salt, followed by the password again
3. md5 the password, followed by a fixed salt, followed by the second hash, followed by parts of the password, but interspersed with zeros
4. hash the password
5. hash the third hash
6. Iterate the previous two steps 50 times, including the password in the hash every seventh time, and salting the hash too, except every third time
7. md5 the result of that loop...

And we have only barely begun.

The Sacred, but Mostly the Profane

fish wrote a client for Yahoo chat, but the protocol was not public. Reverse engineering the login protocol for a native OS X client meant running Etheral in X11 to inspect a Java program running in the OS 9 Classic environment: a remarkable feat, but man, was it slow going. For a long time, connection attempts were met with radio silence and disconnection. Nothing, nothing, nothing....

And then, all at once, Yahoo unleashed a stream of filthy, filthy obscenities. Yessss.

You see, Yahoo was concerned that people might swear on the Internet, so they provided a list of words that the client should filter. But this list might need to be updated dynamically, in case someone on the Internet managed to think up a new word for sex. So rather than build the list into the client, they sent it to you from the server. Right in the first packet. In alphabetical order. Login successful, bitch.

A kind soul has preserved a packet dump from a successful login. Cover your children's eyes and read the box below:

59 43 48 54 00 00 01 00 : 00 00 00 01 00 00 01 7F	YCHT
41 73 6B 46 6F 72 42 6F : 6F 7A 65 C0 80 61 68 6F	AskForBooze;Äaho
6C 65 2C 61 68 6F 6C 65 : 73 2C 61 73 73 68 6F 6C	le,aholes,asshol
65 2C 61 73 73 68 6F 6C : 65 73 2C 61 73 73 77 69	e,assholes,asswi
70 65 2C 62 69 61 74 63 : 68 2C 62 69 74 63 68 2C	pe,biatch,bitch,
62 69 74 63 68 65 73 2C : 62 6C 6F 5F 6A 6F 62 2C	bitches,blo_job,
62 6C 6F 77 5F 6A 6F 62 : 2C 62 6C 6F 77 6A 6F 62	blow_job,blowjob
2C 63 6F 63 6B 73 75 63 : 6B 65 72 2C 63 75 6E 74	,cocksucker,cunt
2C 63 75 6E 74 73 2C 64 : 69 63 6B 68 65 61 64 2C	,cunts,dickhead,
66 75 63 6B 2C 66 75 63 : 6B 65 64 2C 66 75 63 6B	fuck,fucked,fuck
69 6E 67 2C 66 75 63 6B : 6F 66 66 2C 66 75 63 6B	ing,fuckoff,fuck
73 2C 68 61 6E 64 6A 6F : 62 2C 68 61 6E 64 6A 6F	s,handjob,handjo
62 73 2C 6D 6F 74 68 65 : 72 66 75 63 6B 65 72 2C	bs,motherfucker,
6D 6F 74 68 65 72 2D 66 : 75 63 6B 65 72 2C 6D 6F	mother-fucker,mo
74 68 65 72 66 75 63 6B : 65 72 73 2C 6D 75 74 68	therfuckers,muth
61 66 75 63 6B 65 72 2C : 6D 75 74 68 61 66 75 63	afucker,muthafuc
6B 65 72 73 2C 6E 69 67 : 67 61 2C 6E 69 67 67 61	kers,nigga,nigga
73 2C 6E 69 67 67 65 72 : 2C 6E 69 67 67 65 72 73	s,nigger,niggers
2C 70 65 64 6F 66 69 6C : 65 2C 70 65 64 6F 70 68	,pedofile,pedoph
69 6C 65 2C 70 68 61 67 : 2C 70 68 75 63 2C 70 68	ile,phag,phuc,ph
75 63 6B 2C 70 68 75 63 : 6B 65 64 2C 70 68 75 63	uck,phucked,phuc
6B 65 72 2C 73 68 61 74 : 2C 73 68 69 74 2C 73 68	ker,shat,shit,sh
69 74 73 2C 73 68 69 74 : 68 65 61 64 2C 73 68 69	its,shithead,shi
74 74 65 72 2C 73 68 69 : 74 74 69 6E 67 C0 80 54	tter,shitting;ÄT
61 6E 67 6F 62 68 C0 80 : 20 C0 80 30 C0 80 31	angobh;Ä ;Ä0;Ä1

Eat your heart out, George Carlin.

R.I.P. Yahoo chat. You will be remembered as you were: a crazy phuc. Whatever that means. ■

ridiculous_fish is a curious programmer perpetually out of his element. He is channeled by an engineer who currently works at Apple. Read more from fish at ridiculousfish.com

Reprinted with permission of the original author.
First appeared in hn.my/ychat (ridiculousfish.com)

How I Made Porn 20x More Efficient with Python

BY GERGELY KALMAN

PORN IS A big industry. There aren't many sites on the Internet that can rival the traffic of its biggest players.

And juggling this immense traffic is tough. To make things even harder, much of the content served from porn sites is made up of low latency live streams rather than simple static video content. But for all of the challenges involved, rarely have I read about the developers who take them on. So I decided to write about my own experience on the job.

What's the problem?

A few years ago, I was working for the 26th (at the time) most visited website in the world — not just the porn industry: the world.

At the time, the site served up porn streaming requests with the Real Time Messaging protocol (RTMP). More specifically, it used a Flash Media Server (FMS) solution, built by Adobe, to provide users with live streams. The basic process was as follows:

1. The user requests access to some live stream
2. The server replies with an RTMP session playing the desired footage

For a couple reasons, FMS wasn't a good choice for us, starting with its costs, which included the purchasing of both:

1. Windows licenses for every machine on which we ran FMS.
2. ~\$4k FMS-specific licenses, of which we had to purchase several hundred (and more every day) due to our scale.

All of these fees began to rack up. And costs aside, FMS was a lacking product, especially in its functionality (more on this in a bit). So I decided to scrap FMS and write my own RTMP parser from scratch.

In the end, I managed to make our service roughly 20x more efficient.

Getting started

There were two core problems involved: firstly, RTMP and other Adobe protocols and formats were not open (i.e., publically available), which made them hard to work with. How can you reverse or parse files in a format about which you know nothing? Luckily, there were some reversing efforts available in the public sphere (not produced by Adobe, but rather by *osflash.org* who've since taken them down) on which we based our work.

Note: Adobe later released "specifications" which contained no more information than what was already disclosed in the non-Adobe-produced reversing wiki and documents. Their (Adobe's) specifications were of an absurdly low quality and made it near impossible to actually use their libraries. Moreover, the protocol itself seemed intentionally misleading at times.

For example:

1. They used 29-bit integers.
2. They included protocol headers with big endian formatting everywhere — except for a specific (yet unmarked) field, which was little endian.
3. They squeezed data into less space at the cost of computational power when transporting 9k video frames, which made little to no sense, because they were earning back bits or bytes at a time — insignificant gains for such a file size.

And secondly: RTMP is highly session oriented, which made it virtually impossible to multicast an incoming stream. Ideally, if multiple users wanted to watch the same live stream, we could just pass them back pointers to a single session in which that stream is being aired (this would be multicasting). But with RTMP, we had to create an entirely new instance of the stream for every user that wanted access. This was a complete waste.



My solution

With that in mind, I decided to repackage/parse the typical response stream into FLV “tags” (where a “tag” is just some video, audio, or meta data). These FLV tags could travel within the RTMP with little issue.

The benefits of such an approach:

1. We only needed to repackage a stream once (repackaging was a nightmare due to the lack of specifications and protocol quirks outlined above).
2. We could re-use any stream between clients with very few problems by providing them simply with an FLV header, while an internal pointer to FLV tags (along with some sort of offset to indicate where they’re at in the stream) allowed access to the content.

I began development in the language I knew best at the time: C. Over time, this choice became cumbersome; so I started learning Python while porting over my

C code. The development process sped up, but after a few demos, I quickly ran into the problem of exhausting resources. Python’s socket handling was not meant to

handle these types of situations: specifically, in Python we found ourselves making multiple system calls and context switches per action, adding a huge amount of overhead.

Improving performance: mixing Python and C

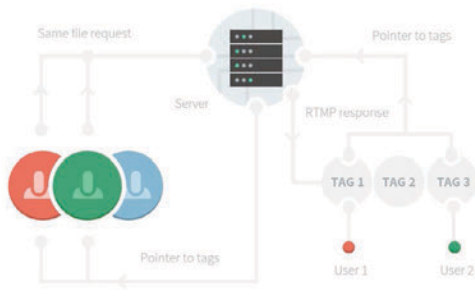
After profiling the code, I chose to move the performance-critical functions into a Python module written entirely in C. This was fairly low-level stuff: specifically, it made use of the kernel’s `epoll` mechanism to provide a logarithmic order-of-growth.

In asynchronous socket programming there are facilities that can provide you with info whether a given socket is readable/writable/error-filled. In the past, developers have used the `select()` system call to get this information, which scales badly. `Poll()` is a better version of `select`, but it’s still not that great, as you have to pass in a bunch of socket descriptors at every call.

Epoll is amazing, as all you have to do is register a socket and the system will remember that distinct socket, handling all the gritty details internally. So there’s no argument-passing overhead with each call. It also scales far better and returns only the sockets that you care about, which is way better than running through a list of 100k socket descriptors to see if they had events with bitmasks (which you need to do if you use the other solutions).

But for the increase in performance, we paid a price: this approach followed a completely different design pattern than before. The site’s previous approach was (if I recall correctly) one monolithic process which blocked on receiving and sending; I was developing an event-driven solution, so I had to refactor the rest of the code as well to fit this new model.

Specifically, in our new approach, we had a main loop, which handled receiving and sending as follows:



1. The received data was passed (as messages) up to the RTMP layer.
2. The RTMP was dissected and FLV tags were extracted.
3. The FLV data was sent to the buffering and multicasting layer, which organized the streams and filled the low-level buffers of the sender.
4. The sender kept a struct for every client, with a last-sent index, and tried to send as much data as possible to the client.

This was a rolling window of data, and included some heuristics to drop frames when the client was too slow to receive. Things worked pretty well.

Systems-level, architectural, and hardware issues

But we ran into another problem: the kernel's context switches were becoming a burden. As a result, we chose to write only every 100 milliseconds, rather than instantaneously. This aggregated the smaller packets and prevented a burst of context switches.

Perhaps a larger problem lied in the realm of server architectures: we needed a load-balancing and failover-capable cluster — losing

users due to server malfunctions is not fun. At first, we went with a separate-director approach, in which a designated “director” would try to create and destroy broadcaster feeds by predicting demand. This failed spectacularly. In fact, everything we tried failed pretty substantially. In the end, we opted for a relatively brute-force approach of sharing

broadcasters among the cluster's nodes randomly, equaling out the traffic.

This worked, but with one drawback: although the general case was handled pretty well, we saw terrible performance when everyone on the site (or a disproportionate number of users) watched a single broadcaster. The good news: this never happens outside a marketing campaign. We implemented a separate cluster to handle this scenario, but in truth we reasoned that jeopardizing the paying user's experience for a marketing effort was senseless — in fact, this wasn't really a genuine scenario (although it would have been nice to handle every imaginable case).

Conclusion

Some statistics from the end-result: Daily traffic on the cluster was about a 100k users at peak (60% load), ~50k on average. I managed two clusters (HUN and US); each of them handled about 40 machines to share the load. The aggregated bandwidth of the clusters was around 50 Gbps, from which they used around 10 Gbps while at peak load. In the end, I managed to push out 10 Gbps/machine easily; theoretically, this number could've gone

as high as 30 Gbps/machine, which translates to about 300k users watching streams concurrently from one server.

The existing FMS cluster contained more than 200 machines, which could've been replaced by my 15 — only 10 of which would do any real work. This gave us roughly a $200/10 = 20\times$ improvement.

Probably my greatest take-away from the project was that I shouldn't let myself be stopped by the prospect of having to learn a new skill set. In particular, Python, transcoding, and object-oriented programming, were all concepts with which I had very sub-professional experience before taking on this project.

That, and that rolling your own solution can pay big. ■

Gergely Kalman is a recruiting engineer with Toptal. With a background in IT-Security, Gergely has worked as Lead Developer for an Alexa Top 50 website serving several million unique visitors each month. He is a diligent and motivated worker who likes to dive in and get things done.

Reprinted with permission of the original author.
First appeared in hn.my/python20 (toptal.com)



MEET MANDRILL

By MailChimp



Mandrill is the fastest way to send transactional, triggered, and personalized emails.
It's also the world's largest species of monkey.

[MANDRILL.COM](https://mandrill.com)

Programmer, Interrupted

By CHRIS PARNIN

I'M WRITING THIS article in an apt state: low-sleep, busy, disorientated, and interrupted. I try all the remedies: Pomodoro, working in coffee shops, headphones, and avoiding work until being distraction free in the late night.

But it is only so long before interruption finds a way to pierce my protective bubble. Like you, *I am programmer, interrupted*. Unfortunately, our understanding of interruption and remedies for them are not too far from homeopathic cures and bloodletting leeches.

But what is the evidence and what can we do about it? Every few months I still see programmers who are asked to not use headphones during work hours or are interrupted by meetings too frequently but have little defense against these claims. I also fear our declining ability to handle these mental workloads and interruptions as we age.

The costs of interruptions have been studied in office environments. An interrupted task is estimated to take twice as long and contain twice as many errors as uninterrupted tasks [hn.my/czerwinski04]. Workers have to work in a fragmented state as 57% of tasks are interrupted [hn.my/mark05].

For programmers, there is less evidence of the effects and prevalence of interruptions. Typically, the number that gets tossed around for getting back into the “zone” is at least 15 minutes after an interruption. Interviews with programmers produce a similar number [hn.my/vansolingen98]. Nevertheless, numerous figures have weighed in: Paul Graham stresses the differences between a maker’s schedule and manager’s schedule. Jason Fried says the office is where we go to get interrupted.

Interruptions of Programmers

Based on an analysis of 10,000 programming sessions recorded from 86 programmers using Eclipse and Visual Studio and a survey of 414 programmers [hn.my/parnin10], we found:

- A programmer is likely to get just one uninterrupted 2-hour session in a day.
- We also looked at some of the ways programmers coped with interruption:
- For most sessions, programmers navigated to several locations to rebuild context before resuming an edit.
- Programmers insert intentional compile errors to force a “road-block” reminder.
- A source diff is seen as a last resort way to recover state but can be cumbersome to review.
- A programmer takes between 10-15 minutes to start editing code after resuming work from an interruption.
- When interrupted during an edit of a method, only 10% of times did a programmer resume work in less than a minute.

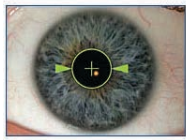


Figure 1

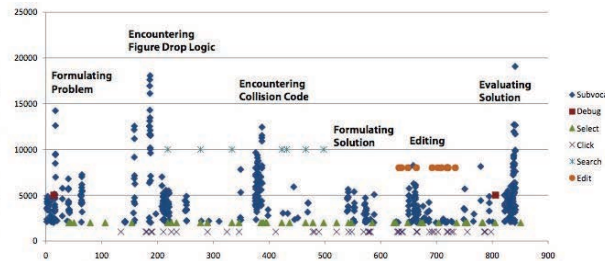
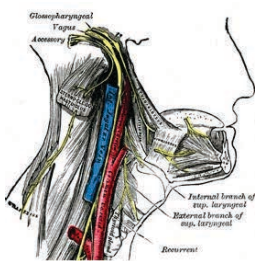
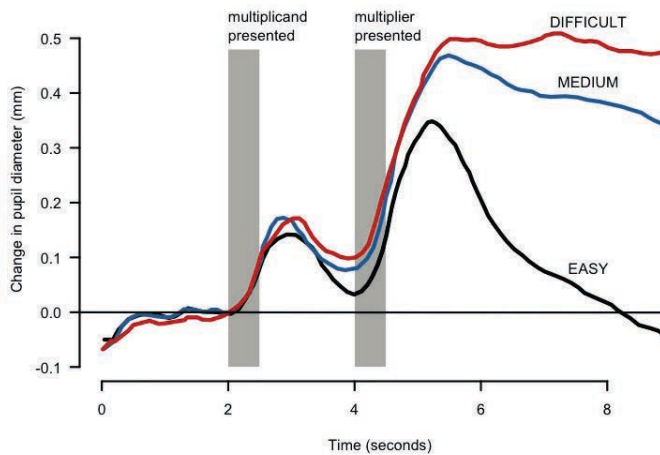


Figure 2

Worst Time to Interrupt a Programmer

Research shows (figure 1) that the worst time to interrupt anyone is when they have the highest memory load. Using neural correlates for memory load, such as pupillometry, studies have shown that interruptions during peak loads cause the biggest disruption [hn.my/iqbal04].

We looked at (figure 2) subvocal utterances during programming tasks to find different levels of memory load during programming tasks [hn.my/parnin11].

If an interrupted person is allowed to suspend their working state or reach a “good break-point,” then the impact of the interruption can be reduced [hn.my/trafton03]. However, programmers often need at least 7 minutes before they transition from a high memory state to a low

memory state [hn.my/iqbal07]. An experiment evaluating in which state a programmer less desired an interruption found these states to be especially problematic [hn.my/fogarty05]:

- During an edit, especially with concurrent edits in multiple locations.
- Navigation and search activities.
- Comprehending data flow and control flow in code.
- IDE window is out of focus.

A Memory-Supported Programming Environment

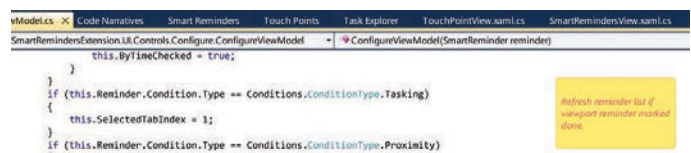
Ultimately, we cannot eliminate interruptions. In some cases interruption may even be beneficial. But we can find ways to reduce the impact on the memory failures that often result from interruption. I introduce some types of memory that get disrupted or heavily burdened during programming and some conceptual aids that can support them.

Prospective Memory

Prospective memory holds reminders to perform future actions in specific circumstances (e.g., to buy milk on the way home from work).

Various studies have described how developers have tried to cope with prospective memory failures. For example, developers often leave TODO comments in the code they are working on [hn.my/storey08]. A drawback of this mechanism is that there is no impetus for viewing these reminders. Instead, to force a prospective prompt, developers may intentionally leave a compile error to ensure they remember to perform a task [hn.my/parnin10]. A problem with compile errors is that they inhibit the ability to switch to another task on the same codebase. Finally, developers also do what other office workers do: leave sticky notes and emails to themselves [hn.my/parnin10].

A smart reminder is a reminder that can be triggered based on conditions such as a teammate checking in code, or proximity to a reminder:



Attentive Memory

Attentive memory holds conscious memories that can be freely attended to.

Some programming tasks require developers to make similar changes across a codebase. For example, if a developer needs to refactor code in order to move a component from one location to another or to update the code to use a new version of an API, then that developer needs to systematically and carefully edit all those locations affected by the desired change. Unfortunately, even a simple change can lead to many complications, requiring the developer to track the status of many locations in the code. Even worse, after an interruption to such a task, the tracked statuses in attentive memory quickly evaporate and the numerous visited and edited locations confound retrieval.

Touch points allow a programmer to track status across many locations in code.



Associative Memory

Associative memory holds a set of non-conscious links between manifestations of co-occurring stimuli.

Developers commonly experience disorientation when navigating to unfamiliar code. The disorientation stems from associative memory failures that arise when a developer must recall information about the places of code they are viewing or what to view next. Researchers believe the lack of rich and stable environmental cues in interface elements, such as document tabs, prevent associative memories from being recalled.

The presence of multiple modalities in a stimulus increases the ability to form an associative memory. In this sense, a modality refers to distinct type of perceptions that is processed by distinct regions of the brain, such as auditory or visual pathways. Examples of different modalities include lexical, spatial, operational, and structural. When

multiple modalities are present in the same stimulus, more pathways are activated, thus increasing the chance of forming an associative memory. In contrast, a monotonous stimulus with a single modality is less likely to form an associative memory.

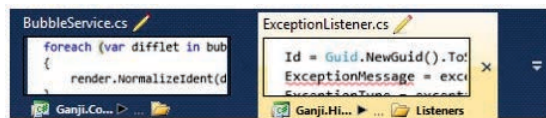
An associative link helps a programmer by situating information of multiple modalities with a program element. In particular, by improving navigating document tabs, which have default configurations that are especially spartan and often just show the name of the document.

Episodic Memory

Episodic memory is the recollection of past events.

Software developers continually encounter new learning experiences about their craft. Retaining and making use of that acquired knowledge requires developers to recollect those experiences from their episodic memory. When recalling from episodic memory, developers commonly experience failures that limit their ability to recall essential details or recollect the key events. For example, a developer may forget the changes they performed for a programming task, or forget details, such as the blog post that was used for implementing a part of the task.

A code narrative is an episodic memory aid that helps a developer recall contextual details and the history of programming activity. Two narrative structures are currently supported: A review mode for high-level recall of events and a share mode for publishing a coding task for others.



Conceptual Memory

Conceptual memory is a continuum between perceptions and abstractions. How does the brain remember such objects as a hammer and such concepts as a tool? The brain first learns basic features of encountered stimuli, such as the wood grains and metal curves of a hammer, and then organizes those features into progressively higher levels of abstraction.

Developers are expected to maintain expertise in their craft throughout their careers. Unfortunately, the path to becoming an expert is not easily walked: For a novice, evidence suggests this can be a 10 year journey [hn.my/chi82]. Furthermore, for experts trying to become experts in new domains, like the desktop developer becoming a web developer, there are many concepts that must be put aside and new ones learned.

Studies examining the difference between an expert and novice find that performance differences arise from differences in brain activity. Not only do experts require less brain activity than novices, they also use different parts of their brains [hn.my/milton07]: Experts use conceptual memory whereas novices use attentive memory. That is, experts are able to exploit abstractions in conceptual memory, whereas novices must hold primitive representations in attentive memory.

Sketchlets (alpha) helps a programmer form and prime concepts by supporting abstraction and reviewing concepts that need to be refreshed.



Future

- fMRI studies of programmers. See preliminary research! [hn.my/prelim]
- Will future programmers take designer nootropics for boosting memory and attention to keep up?
- Can we predict the memory load of using a language feature or performing particular programming tasks?
- What new tool ideas can be derived for programmers?
- What experiments need to be run?

Interested in participating in an experiment or have any ideas? Email me at chris.parnin@gatech.edu ■

Chris Parnin is Phd candidate at Georgia Tech studying software engineering from empirical, hci, and cognitive neuroscience perspectives. He toggles between being a professional software developer and researching them.

Reprinted with permission of the original author.
First appeared in hn.my/interrupted (ninlabs.com)

Sleep: Everything You Need To Know

By MAROUN NAJJAR

WE SPEND ONE third of our lives sleeping — it's crucial for muscle recovery, fact retention and preparing the body to operate at full speed the next day. Sleep is one of the most important things when it comes to day-to-day happiness. But what about students studying late into the night reducing the amount of information they retain? Or athletes sleeping in warm and loud environments missing out on crucial muscle and immune system recovery?

Sleep is a powerful tool that when leveraged the right way can help improve your quality of life tremendously. Many people see sleep as this complex dark place and don't know the first thing about making it better.

A couple months ago, I was one of those people.

I Slept Like Crap

What you see above is my average week of sleep last fall. Yes, I slept 4 hours and 30 minutes on average with less than 90 minutes of REM and Deep. Mind you, this was in the middle of one of my toughest semesters, I was lifting daily, running notably and under a ton of stress.

I felt like complete shit. All. The. Time.

There was not a single point in the day where I wouldn't think "I'm tired." I hated waking up. My lifting stalled. Large coffees with a turbo shot weren't enough to keep me awake so I would rely on two 24oz Monster Energy drinks daily.

Then, something interesting happened. Last January, I started working for a sleep technology startup. As a product manager, my job was to make the best product possible that would help people improve their sleep.

I walked into the office on January 9th and suddenly had access to some of the most knowledgeable people on sleep out there.

So I took the opportunity and I learned as much as I could about sleep. I asked questions and began deciphering what had been one of the most stressful aspects of my life.

Here is what I learned.

Sleep Affects Everything

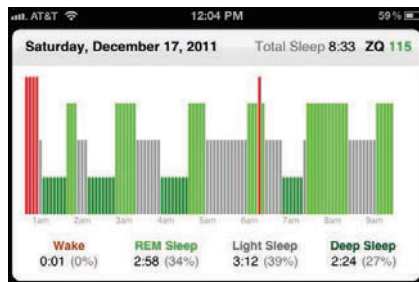
Fact retention, muscle recovery, hormone regulation and alertness. Sleep influences everything.

Let's take someone who just had a horrible night of sleep: Their hormones are going to be all over the place the next day. They will be tempted to eat starchy carbs and be more prone to putting on fat due to carbs.

If you slept like crap, stay away from carbs the next day.

What Exactly is REM and Deep Sleep?

When you sleep you undergo several phases: REM, Light and Deep. Your typical night of sleep looks something like this:



My night of sleep the night I went home after finals... I had some catching up to do.

Deep + Light Sleep:

Think of deep sleep as recovering from the day. Here's what happens:

- You enter Deep Sleep earlier in the night (the later you go to sleep, the less Deep Sleep you get).
- Your neurons slow down reducing the stimuli that comes into your brain, causing your body to release growth hormones that help muscle tissue and immune system recovery.

- If you drink caffeine, any amount will reduce your deep sleep. Keep it to small doses earlier in the day.
- Crucial for fact retention. While in Deep Sleep your mind replays your short-term memories in the hippocampus.

REM Sleep

So you've recovered from the day — awesome. Now let's prepare to get you operating at full speed tomorrow. That's where REM sleep comes in:

- In REM your short-term memories are transferred over to your long-term memory (procedural) area of the brain.
- While Deep increases the amount of facts you remember, REM increases your ability to make connections between those facts, which is especially important for creatives.
- REM sleep affects whether you feel refreshed and focused the next day.

Now that we have a basic understanding of what REM and Deep sleep are, let's take a look at everyone's favorite drug.

Caffeine and Sleep

Caffeine is a wonderful thing; it can help get you through those slow mornings and crank out a ton of work. Plus, coffee is just delicious. Here are two things to keep in mind:

- Caffeine will reduce the amount of deep sleep you get. The more caffeine you have, and the later you have it, the less deep sleep you'll get.

- Caffeine has a half-life of just under six hours (it takes a while to get out of your system).
- If you don't want caffeine to affect your sleep, don't drink it after 3pm.

Napping

What about napping?

- Naps are an awesome way to get some extra rest in the day (our Nap room in the office is in pretty high demand).
- If you're already struggling with sleeping at night, then don't nap.
- The earlier in the day, the better.

Things You Should Focus On To Improve Your Sleep

Here are the four most powerful areas you should focus on to take control of your sleep.

1 Set Up A Sleep Schedule

Consistency, consistency, consistency.

- Go to sleep at the same time and wake up at the same time.
- This is arguably more important than how long you sleep.
- When you stick to a sleep schedule you will feel amazing throughout the day as your circadian rhythm will be set and you'll be able to keep it consistent.
- Use a 40 minute sleep window. For example, I go to sleep anytime between 11:20-12:00 every night and wake up at 7. This ensures I'm hitting 7+ hours a night.

2 Quiet, Dark and Cool

Sleeping in the right atmosphere is crucial to getting a good night's sleep.

- Keep it quiet. Your brain doesn't turn off when you go to sleep. We are still sensitive to noise when we sleep and any sounds can awaken us throwing off our sleep cycles.
- Possibly the best thing I've done for my sleep is wear earbuds. There is nothing that compares to sleeping in absolute silence. Don't get crappy earbuds either; check out Mack's Earbuds. [hn.my/mack]
- Keeping your room cool and drinking cold water right before bed is a great way to stimulate more deep sleep.
- Sleep in the dark. Get dark shades or a tempur-pedic iMask. You don't want to be woken up by the sun. You want your body to sleep for as long as it needs to in a dark and quiet environment.
- Sleeping with socks also helps keep your extremities warm which can increase your deep sleep.

3 Relaxation

It's crucial to learn to relax right before bed. We are constantly in two different states:

Parasympathetic: The state of rest and digest. While in this state your blood pressure and heart rate go down. Your digestion increases, and you are in a state of relaxation.

This is the opposite of the Sympathetic, which is a state of "readiness."

Learn to relax. Create a before bed routine; dim the lights and read a book to get your body used to

"relaxing." Yoga is amazing at getting you in the state of relaxation. Ambient sounds and white noise are also great alternatives.

4 Track It

Just like putting on muscle, losing weight, becoming faster or stronger, you need to track your progress.

Chances are you don't have the slightest idea of how well you sleep. You need to track it to get a picture of what you're working with and then use those measurements to calculate progress.

When I started, I tracked my sleep nightly. I've now gotten to the point where I track it for one full week once a month as a "check-point" to see how I'm doing. If my sleep is bad, then I work to fix it and track it along the way.

Here's what I recommend: Pick one of the methods below and track your sleep for five days. Get a picture of what your sleep looks like and then if you see a need for improvement, test out the methods above.

Methods to Track Your Sleep

Stopwatch (free)

- **Tracks:** Sleep Duration
- **What is it:** A simple stopwatch. Start it before bed and stop it when you wake up.

Sleep101 [sleep101app.com] and other "Sleep Tracking" apps (free)

- **Tracks:** Sleep Duration, Awake vs. Asleep, Time to Fall Asleep, Sleep Efficiency
- **What is it:** A free sleep app that we launched while I was at Zeo. You put it on your bed and it picks up on your movements throughout the night

and determines how deeply you're asleep. I'd recommend this solution because it is free, provides a hefty amount of data and is almost as accurate as wrist devices.

Wrist Devices (\$99-\$150)

- **Tracks:** Sleep Duration, Awake vs. Asleep, Time to Fall Asleep, Sleep Efficiency
- **What is it:** A wristband that you wear while sleeping. It relies on wrist movements to determine how deeply asleep you are. There are several of these on the market like the Jawbone Up, Fitbit, etc.

Zeo [myzeo.com] (\$99)

- **Tracks:** Deep Sleep, Light Sleep, REM Sleep, Sleep Duration, Awake vs. Asleep, Time to Fall Asleep, Overall Sleep Quality.
- **What is it:** A headband that you wear when you go to sleep. It uses brainwaves, eye movement and actigraphy to determine your sleep stages. This is one of the very few things on the market that can measure your actual sleep stages. I'd recommend this for athletes especially.

Sleep affects everything. If you're feeling stressed, exhausted and unfocused, then take a look at your sleep. It's a lot easier to take control of than you might think.

Sleep Better. Live Better. ■

Maroun is a User Experience Researcher at Kicksend and an Engineering student at Northeastern. He's fascinated by health and technology and building great experiences where those two converge.

Reprinted with permission of the original author.
First appeared in hn.my/sleep (medium.com)