Hacking Strength

Matt Might

HACKERMONTHLY

Issue 36 May 2013

Now you can hack on DuckDuckGo

DuckDuckHack

Create instant answer plugins for DuckDuckGo

duckduckhack.com

Curator
Lim Cheng Soon

Contributors
Matt Might
JR Heard
Kris Jordan
Allison Kaptur
Caleb Doxsey
Clay Allsopp
Svarichevsky Mikhail
Sean Gransee

Illustrator
Ben Mounsey

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.

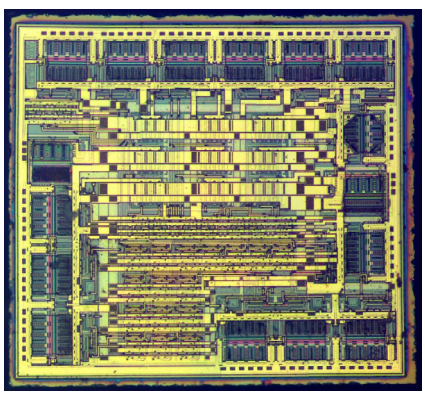
Cover Illustration: Ben Mounsey

Contents

FEATURES

PROGRAMMING

SPECIAL



74HC595 — 8-bit shift register

For links to Hacker News dicussions, visit *hackermonthly.com/issue-36*

Hacking Strength

Gaining Muscle With Least Resistance

By MATT MIGHT

After my daughter was born, I made a commitment to get "in shape."

I lost close to 50 pounds (23 kg) in six months.

Eight inches came off my waist.

Then, after realizing I was too weak to lift my son and his wheelchair in and out of the car, I spent just over a year gaining strength and muscle.

I gained about 35 pounds (16kg).

My waist size increased, but only by an inch.

My strength exploded.

I feel like a different human being.

Having spent the prior three decades of my life mostly out of shape and inert, I found "fitness culture" foreign, and to some extent, I still do.

I exercise three days a week and only for 30 minutes each day.

I only exercise at home.

I did not radically alter my feeding habits (but I did alter them).

In keeping with my "least resistance" philosophy, I created a flexible plan and crafted an environment that left little room for excuses.

Friends, colleagues and family keep asking me how I did it.

I'm describing what I learned, my experience and my fitness journey below.

While my exact path may not be the right fit for everyone, there is value in the philosophy of making exercise as easy and as accessible as possible.

Applying least resistance to gaining muscle

The least-resistance philosophy dictates that you should mold your environment so that the path of least resistance is the path of maximum productivity.

The core principles are eliminating barriers to engaging in productive behavior and erecting barriers to engaging in counter-productive behavior.

Admittedly, this philosophy is tricky to apply to an activity that stresses reaching "maximum resistance."

But, the principles still work.

Going to the gym is a major transaction cost.

Putting on gym clothes is a barrier.

Finding a consecutive block of time to work out induces opportunity costs.

If you set up your equipment at home so that you can walk in, do a set and walk out, you are at the gym whenever you're home.

There are mixed opinions as to whether concentrated or dispersed exercise leads to faster muscle growth.

Whatever the case may be, it is clearly the case that both styles work.

The key to success is choosing compact equipment with low per-use overhead that can still target any muscle group with an arbitrary load.

If you make exercise easy and accessible, you are more likely to exercise.

> **“Choose compact equipment with low per-use overhead that can still target any muscle group with an arbitrary load.”**

Why muscle grows

Before I did anything, I did some research.

I looked for consistent advice backed by the scientific literature.

A few core principles emerged:

1. To gain weight, you must eat more calories than you burn.
2. When you gain weight, some will be fat and some will be muscle.
3. To gain muscle, you must eat protein.
4. To boost the ratio of muscle to fat gain, you must exercise.
5. To gain muscle and strength, you must engage in resistance exercise.
6. To prioritize strength over size gains, aim for 2-6 repetitions per exercise, where failure occurs on the last repetition.
7. To prioritize size over strength gains, aim for 8-12 repetitions per exercise, where failure occurs on the last repetition.
8. To maximize gains per unit of exercise, do not work the same muscle group two days in a row.

There are a few corollaries to these principles:

1. To continue gaining strength, you must increase resistance.
2. To optimize strength and size gains, vary repetitions over time.
3. To improve body fat percentage and gain weight, you must cycle through phases of gaining muscle and losing fat.
4. Rest thoroughly.

A word on the scientific literature

The scientific literature on fitness varies in quality and, at times, misleads.

There are many seemingly contradictory findings about the (non-?)importance of protein, resistance exercise, supplements, etc.

Two problems plague the literature:

1. Sample sizes too small to draw meaningful conclusions.
2. Subsequent misinterpretations of statistical significance.

There are articles cited as evidence that “X does not improve muscle growth,” where upon further inspection, the data might actually show a positive correlation between X and muscle growth.

Because the finding was not “statistically significant,” it was misinterpreted as “there is no correlation.”

A lack of “statistical significance” does not mean there is no correlation.

A lack of “statistical significance” means the sample size was too small to render judgment at the desired level of confidence (usually chosen by the experimenter somewhat arbitrarily to be 90%, 95% or 99%).

The presence of a correlation but a lack of statistical significance should be interpreted as “preliminary research is promising, but more experimentation is needed to render judgment.”

The bottom line is that you must interpret the data yourself.

Form the habit first

Initially, developing an exercise habit is more important than the specific exercise you do.

Do push-ups, weights, sit-ups, jogging, cycling, pull-ups or whatever.

At first, it doesn't matter.

You need the habit so that when injury strikes or travel interrupts, you know you'll go back to your routine.

Do it on three non-consecutive days a week for about half an hour.

Bodyweight exercises are good enough for building the habit because they don't need any equipment. The transaction cost to exercise is near zero.

A detachable pull-up bar is a cheap and effective way to expand your options with bodyweight exercises.

Diet

Gaining weight, whether fat or muscle, requires over-eating.

To gain a pound of muscle or fat takes over-eating by about 3500 calories.

It's challenging to gain even a 50/50 ratio of muscle/fat, so be sure to supply your body with protein on all days to capture the lasting effects of resistance training as a stimulus to muscle growth.

There's a mixture of opinions as to whether the timing of protein ingestion matters, and to what extent it matters if it matters.

There are claims and counter-claims that the body can absorb only up to 30g of protein "at a time."

I never resolved these debates.

I recommend eating protein with every meal, and before and after exercise.

Eating six meals stretched across the day probably does improve muscle growth at the margins, but it is certainly inconvenient and, in my experience, unnecessary.

Recommended amounts of protein vary from 0.8 grams per kilogram of bodyweight per day to 2 grams per kilogram.

I aimed for roughly 1.5 grams per kilogram, but not stringently.

Whey protein makes it easy to hit the target you set:

- I use hydrolized whey, since it is "pre-digested." It absorbs rapidly and avoids some of the socially problematic "side effects" of high-protein diets.
- Whey is a complete protein: it contains every essential amino acid, and beyond that, it is an abundant source of the branch-chain amino acids critical to muscle growth.

A blender bottle avoids the need for a blender (thereby lowering transaction costs to consuming protein), and makes a shake with water or milk. It takes about a minute to make a shake, drink it and rinse out the bottle.

MyFitnessPal makes it easy to track calories and hit a 500 - 1000 calorie surplus on workout days, and a 200-500 calories surplus on rest days.

Track your weight carefully, and adjust your surplus as needed.

An RMR calculator can estimate daily caloric expenditure.

Routines

Since rest is important to growth and healing, lifting on consecutive days is sub-optimal from a growth per unit of exercise perspective.

It also increases the likelihood of injury.

There are routines that lift on consecutive days. These achieve rest by hitting strictly different muscles on consecutive days.

Find the split that works for you.

If you're looking for a starting point, BodyBuilding.com's workout planning page has a variety of plans based on specific goals.

In my current routine, each day has a different upper-body theme: arms; shoulders and back; and chest. I do a few leg exercises on each lifting day.

I saw good results on just 3 days a week.

I'll consider a new routine if I plateau and still need more strength.

While gaining weight, I aim for about 20 sets of 8-12 reps to failure.

While cutting fat, I aim for about 15 sets of 4-8 reps to failure.

Listen to your body.

If it can handle more sets, do more sets. If you're sore days letter, do less.

On each lifting day, pick an assortment of exercises that target the muscle group for that day, with some variation week to week.

Dumbbells first

Dumbbells are a great way to tiptoe into resistance training, because they're a versatile piece of resistance equipment, and they're relatively safe.

From a least resistance perspective, their real strength, however, is their compactness and low transaction cost.

That ease of use is why I moved to dumbbells after bodyweight.

Anyone can fit dumbbells in their home or office.

I began with a simple, 40-Pound adjustable dumbbell set. But these are poorly suited to a least resistance approach because it takes about two minutes to unscrew all four caps and change the plates each time.

And, my small workout room filled with plates as I gained strength.

I borrowed a PowerBlock Elite set from a friend:

To select a new weight on these, you pull and re-insert a two-prong peg on the side. It's a much lower transaction cost than cap-and-plate dumbbells, but you do have to crouch down to face the side and get both pegs to line up. It takes about 10-20 seconds to adjust each one.

Aiming to entirely eliminate that per-lift set-up time, I bought two SelectTech 1090 adjustable dumbbells.

With the flip of the dials, these select between 10 and 90 pounds per hand in 2.5 pound increments.

I wish I'd had these at the start instead of the cumbersome cap and plates.

To reduce the transaction cost to engaging in exercise, I mounted them on the SelectTech dumbbell stand in a room about the size of a walk-in closet. When they were on the floor, the cost of engaging in every dumbbell exercise was an implicit deadlift.

After a month of dumbbell exercises, I added an adjustable bench to my setup to increase the range of exercises I could do. It takes about 20 seconds to set up and begin any exercise.

My favorite dumbbell exercises are:

- Flat chest press.
- Seated overhead shoulder press.
- Chest fly.
- Hammer curls.
- Overhead tricep extension.
- Lateral raises.
- Upright rows.
- Weighted lunges.
- Weighted calf raises.

Of course, there are dozens more you can do with dumbbells. BodyBuilding.com has 123 different dumbbell exercises. Try them all (or the ones that look fun to you).

Safety

Even with dumbbells, I was able to put loads on my back that felt uncomfortable within about three months.

As a precaution, I bought a lifting belt to stabilize my core on these lifts.

I also found it easy to put loads on my wrists that coupled with my propensity for typing all day started to cause some pain.

Padded lifting gloves with wrist support helped keep my wrists in a stable, safe position and alleviated wrist pain.

It's helpful to schedule at least one session with a personal trainer to review proper form. You can only learn so much from YouTube videos.

Pressing

About five months after I started lifting, I hit 90 pounds per hand on the flat bench press (up from 25 per hand at the start).

To increase resistance on the bench press, I had to buy a pressing system.

I considered a traditional barbell system.

While I enjoy using barbells in hotel gyms, safety was my prime concern, and I had no one at home to spot me with a traditional barbell.

Eventually, I picked the Marcy PRO Compact Strength Trainer.

This system is small and it fits in the same small room as my dumbbells.

Critically, it is impossible to crush yourself with this system.

From a least resistance perspective, the per-lift transaction cost is very low after the weights are adjusted for the first lift of the day.

It accepts standard olympic plates up to 500 pounds, so it should be good for a lifetime of lifting needs for the average person.

It supports performing many compound lifts safely and quickly.

It won't engage the stabilizers as well as a traditional barbell, but I was willing to make the sacrifice for safety and efficiency.

The relative position of the handles and the plates does create a 3:2 mechanical advantage, e.g., 300 pounds of plates yields 200 pounds of resistance.

If you're willing to tolerate some additional transaction cost and have a workout room larger than a walk-in closet, you can lift safely with a barbell and power rack, and that will engage stabilizers.

Cutting

There is no way to avoid gaining some fat while building muscle.

As a result, it will be necessary to periodically "cut" fat.

Rest days are critical while cutting and injuries take longer to heal.

Strength gains (except for nervous system adaptations) are unlikely during cutting, so the goal is to maintain rather than gain strength.

If your strength is stable (or decreases only slightly) during a cut, you are preserving muscle and cutting fat.

If your strength is dropping quickly, you may be cutting too fast.

1.5 pounds per week is widely recommended as a safe amount to cut.

Be sure to continue protein supplementation so that the body does not have to break down muscle to meet its nutritional demands.

I've found no harm in aerobic exercise on non-lifting days while cutting.

Supplements

There are countless supplements available that purport to help.

Supplements have the biggest effect when they correct deficiencies.

I've experimented with many, but only a handful had a noticeable impact.

Before trying anything, I recommend checking it out on examine.com first. The site has thoroughly reviewed the scientific literature on almost every supplement you'll encounter.

Whey protein

Whey protein does not have an immediate impact on strength, but it does make it easy to hit protein targets and maintain steady gains.

It is notable for its fast rate of absorption by the body, and its high concentration of amino acids critical to muscle protein synthesis.

These days, I strictly prefer hydrolyzed whey, since it absorbs rapidly and has no socially problematic "side effects."

I did have great results early on with a whey isolate blend. But, I had to take it with psyllium husk fiber every time.

Creatine

In terms of impact, creatine stands out.

With one teaspoon a day, I saw exceptionally rapid gains in strength for several weeks.

Discontinuing creatine did not cause a reversal in strength.

Creatine does cause water retention, and hydration is critical to avoid cramps.

Discontinuing creatine will also end the water retention, resulting in temporarily rapid "weight" loss.

Brain-chain amino acids (BCAAs)

After creatine, branch chain amino acids stand out as useful for blunting hunger and fueling muscles while cutting or fasting.

Small amounts of BCAAs (up to 20g) do not seem to interfere with the benefits of fasting.

There is no need to take BCAAs simultaneously with whey protein, since whey protein contains all three BCAAs.

If you take BCAAs, I recommend the BCAA capsules , since the powder tastes wretched.

Casein protein

Casein protein is a complete protein useful for its uniquely slow rate of absorption.

I don't use it currently, but in the past I used it after my last workout or before bed to provide a steady trickle of protein all night long.

I have also used it to dampen hunger right before a fast.

I discontinued its use not because it was ineffective, but because I felt that micromanaging my proteins at this level was cutting against the least resistance philosophy.

To simplify, I use only whey for protein supplementation.

Vitamin D

Programmers, by nature, tend to have lower sun exposure and lower levels of vitamin D.

Vitamin D deficiency can cause low testosterone.

Modest supplementation with vitamin D can correct that and raise testosterone. Or, you can go outside more often.

Dealing with injury

Good form and adequate rest will help avoid injury.

But, when an injury happens during a workout, end the workout. Exercise on that body part is suspended until recovery.

It's tempting to "power through" an injury.

Just don't do it.

If treated quickly, minor injuries will heal in a day or two.

Taking an entire week off from exercise every couple months also gives your body a chance to fully recover from accumulated stress.

Measuring progress

Keep a spreadsheet to track 1, 5, 8, 10 and 12 rep maximums.

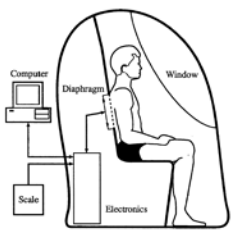
For each exercise, record the weight at which the muscle fails for the given number of reps.

Track maximums to ensure progressive overload.

If you hit 12 reps, it's time to increase the weight.

Track values like cholesterol, blood pressure, blood glucose and triglycerides annually or semi-annually, as these values may rise while gaining weight.

I was able to track my body fat percentage using a "bod pod" at the university health science center:



Tracking body fat percentage is important because you can compute how much you're gaining as fat and how much as lean mass.

According to bod pod measurements, about 14 pounds of my weight gain over my first year of lifting was lean mass.

Close to 20 pounds of what I gained was fat.

I've carefully dropped 10 pounds since then, losing one pound of lean mass and nine pounds of fat in the process.

The next time I start gaining, I can safely lower my caloric surplus.

My 10-rep bench press went from 50 pounds to 220 pounds in one year.

I had 3x-5x improvements in lifts across the board.

Due to diminishing returns, I will not be able to add 14 pounds (or even 10 pounds) of muscle and that much strength over the next year.

Initial gains are much easier, and I was starting from a very blank slate.

But, at this point, it doesn't matter.

I can already lift my son and his wheelchair up a flight of stairs. ■

Matt Might is a professor of Computer Science at the University of Utah. His research interests include programming language design, static analysis and compiler optimization. He blogs at *matt.might.net/articles* and tweets from @mattmight

Reprinted with permission of the original author. First appeared in *hn.my/strength* (might.net)

Illustration by Ben Mounsey.

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

Getting Started With Clojure

By JR HEARD

I'm about to try to teach a bunch of people (primarily Python devs running OS X) how to use Clojure, and I'm not satisfied with any of the currently existing documentation on how to get up and running from scratch. When I was going through all this myself a few months back, there was a weird period of a good few weeks when I had basically no mental map of the Clojure ecosystem and had no idea how to assemble one.

My goal is to create the resource I wish I had six months ago. I'll assume that you're running on OS X and have a non-zero amount of programming experience.

The Clojure Book

Your first step should be to buy and begin reading Clojure Programming [clojurebook.com]. There's another book called (confusingly enough) Programming Clojure, and I can't vouch for whether it's better or worse, but I used Clojure Programming and liked it very much, so it's what I recommend. It's written by people whose names you're going to get used to seeing everywhere as you explore the Clojure ecosystem; all the main figures in the Clojure community seem to be inhumanly prolific.

Let's Get Started

Now, let's start getting your environment assembled. Get Homebrew , "the missing package manager for OS X," if you don't have it already, and then run:

```
brew install leiningen
```

Congratulations, now you have Leiningen! (Make sure you ended up with version 2.0 or greater — you can check that by running `lein --version`.)

So what the hell is Leiningen?

Leiningen's the main tool you'll be using for:

- starting up a REPL
- downloading+installing libraries
- running your programs
- starting a server to run the webapps you've written

Go ahead and run `lein repl`. You've now got a working Clojure REPL! In addition, if you run that command from the top-level directory of one of your Clojure projects, it'll deal with wiring up classpaths and whatnot so that you'll be able to import and play around with your project's code and the libraries that it depends on. We'll get to that later. Right now, let's create a skeleton project for us to play around with by running:

```
lein new foo
```

When that's done, `cd` into `foo` and you'll see that it's already got some files and directories:

```
[jrheard@jrheard-air:~/dev/foo] $ ll
total 16
-rw-r--r--  1 jrheard  Jan  5 15:17 README.md
-rw-r--r--  1 jrheard  Jan  5 15:17 project.clj
drwxr-xr-x  3 jrheard  Jan  5 15:17 src
drwxr-xr-x  3 jrheard  Jan  5 15:17 test
```

Whenever you write a Clojure library/program/anything, your source code will live in the `src` directory and your tests will live in the `test` directory. Straightforward enough. Let's take a look around in `src`:

```
[jrheard@jrheard-air:~/dev/foo] $ cat src/foo/
core.clj
(ns foo.core)

(defn foo
  "I don't do a whole lot."
  [x]
  (println x "Hello, World!"))
```

Looks like Leiningen's already created a file called `src/foo/core.clj`. It's a Clojure program that defines a namespace called `foo.core` and then declares that that namespace contains a function called `foo`. Let's check it out. Start up a repl with `lein repl` and poke around. Remember when I mentioned earlier that leiningen takes care of setting up your classpath and associated goop such that you're able to access your project's code from the REPL? Check this out:

```
user=> (use 'foo.core)
nil
user=> foo
#<core$foo foo.core$foo@6ad591a6>
user=> (foo "jrheard")
jrheard Hello, World!
nil
```

Awesome, we were able to import our code and run it. The `use` function basically serves the same purpose as `from foo.core import *` would in Python, and its use in source code is similarly discouraged for the same reasons that `import *` is discouraged. Like `import *`, though, It's pretty useful to have when you're poking around in the REPL.

So that's cool: we've created a project, it's got code in it, and we've found out how to start up a working REPL that can play around with that code. Bullet point 1: accomplished.

Let's take a look at the second bullet point.

Downloading and installing libraries

You're probably used to getting your libraries by running something from the command-line, e.g. `pip install this_great_library_i_found`, which would download the specified library and install it either globally or within your current virtualenv. Things work a little bit differently in Clojure.

First, you've got to find a library that looks useful. The Clojure Toolbox [clojure-toolbox.com] is a fantastic tool for this, and is the best such resource I've found. Let's choose a library to play around with: making HTTP requests is fun; let's go down to the "HTTP Clients" section and see what our options are. Looks like we've got to pick between clj-http [hn.my/cljhttp] and http.async.client [hn.my/httpasync], but how do we choose?

Currently, my favorite way of deciding between competing libraries is: pull up their respective github repos, compare the number of stars+forks, and give bonus points to any libraries that have commits from within the past month or two. Not exactly scientific, but it has served me well so far as a good proxy for the strength of the library's community/influence/adoption. As of this writing, clj-http has 242 stars to http.async.client's 127, so let's pick clj-http.

So…how do we get it?

Let's go to clj-http's github repo. Check out how the README's installation section has this block of code:

```
[clj-http "0.6.3"]
```

That's the information we need: it's a Clojure vector with two items, the first of which is the name of the library, and the second of which identifies the most up-to-date stable version available. We're going to add this to our `project.clj`, which you saw earlier when we looked at the contents of the `foo` directory. Open up `project.clj`, it'll look like this:

```
(defproject foo "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/
epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.4.0"]])
```

Note the `:dependencies` section; it's a Clojure vector containing one item, and that item is itself a Clojure vector containing two items. This vector indicates to Leiningen that we want our project to run on version 1.4.0 of Clojure. Fair enough; now let's add the clj-http vector we saw earlier. Our `project.clj` should now look like this:

```
(defproject foo "0.1.0-SNAPSHOT"
  :description "FIXME: write description"
  :url "http://example.com/FIXME"
  :license {:name "Eclipse Public License"
            :url "http://www.eclipse.org/legal/
epl-v10.html"}
  :dependencies [[org.clojure/clojure "1.4.0"]
                 [clj-http "0.6.3"]])
```

And that's it! We've now specified to Leiningen that we want the clj-http library, and which version we need. Let's try it out. Start a REPL with `lein repl`, and let's play around with our fancy new library. Notice that Leiningen will first download clj-http before starting up the REPL. That's because it first runs `lein deps` behind the scenes any time you ask it to do basically anything, and that causes it to scan your `project.clj` and make sure that it's already fetched all the dependencies you've asked it to.

Okay, back to our REPL session. Looks like clj-http's github repo's README suggests that you require it in the REPL by running

```
(require '[clj-http.client :as client])
```

So let's do that. It's the same thing as `from clj.http import client` in Python (as opposed to `from clj.http.client import *`, which is again what the `use` function does.)

```
user=> (require '[clj-http.client :as client])
nil
user=> (client/get "http://www.yelp.com")
;; a big huge blob of data pops out!
```

Okay, wow, looks like that worked! That's sort of hard to read. You'll notice that the big huge blob of data ends with a "}", which is a hint that it might be a Clojure map. Let's try poking at it:

```
user=> (def resp (client/get "http://www.yelp.
com"))
#'user/resp
user=> (type resp)
clojure.lang.PersistentArrayMap
user=> (keys resp)
(:cookies :trace-redirects :request-time :status
:headers :body)
user=> (:status resp)
200
user=> (:headers resp)
{"server" "Apache", "content-encoding" "gzip",
"x-proxied" "lb2",
"content-type" "text/html; charset=UTF-8",
"date" "Sun, 06 Jan 2013 00:02:58 GMT",
"cache-control" "private", "vary" "Accept-
Encoding,User-Agent",
"transfer-encoding" "chunked", "x-node" "wsgi,
web40, www_all",
"x-mode" "ro", "connection" "close"}
```

And there you have it: we've found an HTTP client library, downloaded it, and figured out how to use it interactively in the REPL!

It took me a while to figure this all out. After beating my head against a wall for a day, I eventually had to jump into the *#clojure* IRC channel and plead for help. Now you don't have to!

Putting it all together

Let's finish up by figuring out how to actually run a Clojure program. Let's try a good old `lein run`:

```
[jrheard@jrheard-air:~/dev/foo] $ lein run

No :main namespace specified in project.clj.
```

Okay, that didn't work. Referring back to the Leiningen tutorial mentioned earlier and doing a search for `:main`, we see that you can define a :main key in your `project.clj` definition that specifies the namespace that `lein run` will run, and that said namespace has to contain a `-main` function, which serves as the entry point into your program.

Let's add this line to our `project.clj` spec:

```
:main foo.core
```

And finally let's modify `src/foo/core.clj` so that it looks like this:

```
(ns foo.core
  (:require [clj-http.client :as client]))

(defn -main
  "Prints the first 50 characters of the HTML source of
yelp.com."
  [& args]
  (println (apply str
                  (take 50
                        (:body (client/get "http://www.
yelp.com"))))))
```

Here we go — let's try it out with `lein run`!

```
[jrheard@jrheard-air:~/dev/foo] $ lein run
Compiling foo.core
<!DOCTYPE HTML>

<!--[if lt IE 7 ]> <html xmlns:fb
```

It works!

That's it for now! You now have a working REPL to play around with, the ability to install and use libraries, the knowledge to give your programs access to those libraries and run them, and a really good book that'll take you through everything else you need to know about the Clojure language.

The reason I had to write this

Clojure's still a pretty young language. The community is extremely small relative to e.g. Python's, and although the core language's API is (I'm told) remarkably stable, a lot of the tools around it are new and in a state of rapid change. Add on top of that the fact that most of the up-to-date documentation you'll find has poor SEO — to the degree that a lot of your Google searches will turn up documentation on *richhickey.github.com* that's years out of date and deprecated — and you'll find that getting started from scratch can be a little tricky.

I hope that this article has helped save you the few weeks of bewilderment that I went through when I was getting started. I promise that the joy of actually programming in Clojure is well worth putting up with these growing pains. ■

JR Heard dreams about computer programs. He lives in foggy San Francisco, where he writes Python code for Yelp, which helps millions of people across the world find great local businesses. He has some pamphlets about pure functions that he'd like to give to you.

Reprinted with permission of the original author. First appeared in *hn.my/startclojure* (jrheard.tumblr.com)

Hacking on HTTP from the Command-Line

9 Uses for cURL Worth Knowing

By KRIS JORDAN

Working with HTTP from the command-line is a valuable skill for HTTP architects and API designers. The cURL library [curl.haxx.se] and `curl` command give you the ability to design a Request, put it on the pipe, and explore the Response. The downside to the power of `curl` is how much breadth its options cover. Running `curl --help` spits out 150 different flags and options. This article demonstrates nine basic, real-world applications of `curl`.

In this tutorial we'll use the httpkit echo service [echo.httpkit.com] as our end point. The echo server's Response is a JSON representation of the HTTP request it receives.

1 Make a Request

Let's start with the simplest `curl` command possible.

Request

```
curl http://echo.httpkit.com
```

Response

```
{
  "method": "GET",
  "uri": "/",
  "path": {
    "name": "/",
    "query": "",
    "params": {}
  },
  "headers": {
    "host": "echo.httpkit.com",
    "user-agent": "curl/7.24.0 ...",
    "accept": "*/*"
  },
  "body": null,
  "ip": "28.169.144.35",
  "powered-by": "http://httpkit.com",
  "docs": "http://httpkit.com/echo"
}
```

Just like that we have used `curl` to make an HTTP Request. The method, or "verb," `curl` uses by default is `GET`. The resource, or "noun," we are requesting is addressed by the URL pointing to the httpkit echo service, *http://echo.httpkit.com*

You can add path and query string parameters right to the URL.

Request

```
curl http://echo.httpkit.com/path?query=string
```

Response

```
{ ...
  "uri": "/path?query=string",
  "path": {
    "name": "/path",
    "query": "?query=string",
    "params": {
      "query": "string"
    }
  }, ...
}
```

2 Set the Request Method

The `curl` default HTTP method, `GET`, can be set to any method you would like using the `-X` option. The usual suspects `POST`, `PUT`, `DELETE`, and even custom methods, can be specified.

Request

```
curl -X POST echo.httpkit.com
```

Response

```
{
    "method": "POST",
    ...
}
```

As you can see, the `http://` protocol prefix can be dropped with `curl` because it is assumed by default. Let's give `DELETE` a try, too.

Request

```
curl -X DELETE echo.httpkit.com
```

Response

```
{
    "method": "DELETE",
    ...
}
```

3 Set Request Headers

Request headers allow clients to provide servers with meta information about such things as authorization, capabilities, and body content-type. OAuth2 uses an `Authorization` header to pass access tokens, for example. Custom headers are set in `curl` using the `-H` option.

Request

```
curl -H "Authorization: OAuth 2c4419d1aabeec" \
     http://echo.httpkit.com
```

Response

```
{...
"headers": {
    "host": "echo.httpkit.com",
    "authorization": "OAuth 2c4419d1aabeec",
  ...},
...}
```

Multiple headers can be set by using the `-H` option multiple times.

Request

```
curl -H "Accept: application/json" \
     -H "Authorization: OAuth 2c3455d1aeffc" \
     http://echo.httpkit.com
```

Response

```
{ ...
  "headers": { ...
    "host": "echo.httpkit.com",
    "accept": "application/json",
    "authorization": "OAuth 2c3455d1aeffc"
   }, ...
}
```

4 Send a Request Body

Many popular HTTP APIs today `POST` and `PUT` resources using `application/json` or `application/xml` rather than in an HTML form data. Let's try `PUT`ing some JSON data to the server.

Request

```
curl -X PUT \
     -H 'Content-Type: application/json' \
     -d '{"firstName":"Kris",
"lastName":"Jordan"}'
     echo.httpkit.com
```

Response

```
{
   "method": "PUT", ...
   "headers": { ...
     "content-type": "application/json",
     "content-length": "40"
   },
   "body": "{\"firstName\":\"Kris\",\"lastName\":
\"Jordan\"}",
   ...
 }
```

5 Use a File as a Request Body

Escaping JSON/XML at the command line can be a pain and sometimes the body payloads are large files. Luckily, cURL's `@readfile` macro makes it easy to read in the contents of a file. If we had the above example's JSON in a file named `example.json` we could have run it like this, instead:

Request

```
curl -X PUT \
     -H 'Content-Type: application/json' \
     -d @example.json
     echo.httpkit.com
```

6 POST HTML Form Data

Being able to set a custom method, like `POST`, is of little use if we can't also send a request body with data. Perhaps we are testing the submission of an HTML form. Using the `-d` option, we can specify URL encoded field names and values.

Request

```
curl -d "firstName=Kris" \
     -d "lastName=Jordan" \
     echo.httpkit.com
```

Response

```
{
  "method": "POST", ...
  "headers": {
    "content-length": "30",
    "content-type":"application/x-www-form-
urlencoded"
  },
  "body": "firstName=Kris&lastName=Jordan", ...
}
```

Notice the method is `POST` even though we did not specify it. When `curl` sees form field data it assumes `POST`. You can override the method using the `-X` flag discussed above. The "Content-Type" header is also automatically set to "application/x-www-form-urlencoded" so that the web server knows how to parse the content. Finally, the request body is composed by URL encoding each of the form fields.

7 POST HTML Multipart / File Forms

What about HTML forms with file uploads? As you know from writing HTML file upload form, these use a `multipart/form-data` Content-Type, with the `enctype` attribute in HTML. In cURL we can pair the `-F` option and the `@readFile` macro covered above.

Request

```
curl -F "firstName=Kris" \
     -F "publicKey=@idrsa.pub;type=text/plain" \
     echo.httpkit.com
```

Response

```
{
  "method": "POST",
  ...
  "headers": {
    "content-length": "697",
    "content-type": "multipart/form-data;
    boundary=------------------488327019409",
    ... },
  "body": "--------------------488327019409\r\n
           Content-Disposition: form-data;
           name=\"firstName\"\r\n\r\n
           Kris\r\n
           --------------------488327019409\r\n
           Content-Disposition: form-data;
           name=\"publicKey\";
           filename=\"id_rsa.pub\"\r\n
           Content-Type: text/plain\r\n\r\n
           ssh-rsa AAAAB3NzaC1yc2EAAAABI-
wAAAQEAkq1lZYUOJH2
           ... more [a-zA-Z0-9]* ...
           naZXJw== krisjordan@gmail.com\n\r\n
           --------------------488327019409
           --\r\n",
...}
```

Like with the `-d` flag, using `-F curl` will automatically default to the `POST` method, the `multipart/form-data` content-type header, calculate length, and compose the multipart body for you. Notice how the `@readFile` macro will read the contents of a file into any string; it's not just a standalone operator. The ";text/plain" specifies the MIME content-type of the file. Left unspecified, `curl` will attempt to sniff the content-type for you.

8 Test Virtual Hosts, Avoid DNS

Testing a virtual host or a caching proxy before modifying DNS and without overriding hosts is useful on occasion. With cURL, just point the request at your host's IP address and override the default `Host` header cURL sets up.

Request

```
curl -H "Host: google.com" 50.112.251.120
```

Response

```
{
  "method": "GET", ...
  "headers": {
    "host": "google.com", ...
  }, ...
}
```

9 View Response Headers

APIs are increasingly making use of response headers to provide information on authorization, rate limiting, caching, etc. With cURL you can view the headers and the body using the `-i` flag.

Request

```
curl -i echo.httpkit.com
```

Response

```
HTTP/1.1 200 OK
Server: nginx/1.1.19
Date: Wed, 29 Aug 2012 04:18:19 GMT
Content-Type: application/json; charset=utf-8
Content-Length: 391
Connection: keep-alive
X-Powered-By: http://httpkit.com

{
  "method": "GET",
  "uri": "/", ...
}
```

■

Kris Jordan is a Founder and the Technology Director of *NewMediaCampaigns.com*, an interactive agency out of North Carolina. He is the creator of Wiretap [httpkit.com/wiretap], a hosted HTTP proxy for debugging REST API interactions. Contact him anytime at @KrisJordan

Reprinted with permission of the original author.
First appeared in *hn.my/curl* (httpkit.com)

There's No Magic: Virtualenv Edition

By ALLISON KAPTUR

Photo: Jürgen Schiller Garcia [flickr.com/photos/schillergarcia/2934384422]

The more programming I do, the more often I find myself thinking, "Ah, that's not magic." I had one of these moments recently when dealing with a python virtual environment created by virtualenv [virtualenv.org]. Virtualenv creates a sandboxed python environment with its own installation directories, separate from the system python and other virtual environments on your machine. This makes it a great way to test on multiple versions of python or to explore a new package that could break other things you care about.

How does it work? Well, the "magic" works like this:

- Create a virtual environment with `virtualenv my_env`
- Chant the magic incantation `source bin/activate`
- Watch as your previously-failing installation of pygame goes smoothly!
- To "turn off" your virtualenv, the magic incantation is `deactivate`

Of course, it's not actually magic. Virtualenv is a fairly simple (though clever) bash script that does only a couple of things. You don't have to understand much bash scripting to see what's going on. In fact, if you only know python, I'll teach you all the bash you need to understand virtualenv right now.

I'm going to skip the actual creation of a virtual environment, and just focus on what happens when you activate and deactivate that environment. If you'd like to play along, `pip install virtualenv`, create a new virtual environment with `virtualenv testenv`, and then cd into the testenv/ directory that was created. (Don't run `source bin/activate` just yet.)

First, let's look at that incantation, `source bin/activate`. What's going on here? `source` is a bash command that runs a file, the same way you'd use `import` to run your python module. `bin/activate` is the bash script being run.

One other detail of `source` will be important. `source` runs the file provided in your current shell, not in a subshell. Thus it keeps the variables it creates or modifies around after the file is done executing. Since (almost) all that virtualenv does is modify environmental variables, this matters.

OK, now let's look at `bin/activate`. Fire up the activate file in your favorite text editor.

The first thing to notice is that it's only ~80 lines! Cool, we can handle this.

(The `activate` script is generated automatically by the virtualenv installation, and has some system-specific parameters, so your copy may be slightly different from mine..)

The first thing we find is a comment:

```
# This file must be used with
# "source bin/activate"
# *from bash* you cannot run it
# directly
```

We already know why this is true: it's because of the behavior of `source` that we just learned. Running a bash file directly (e.g., calling `activate` from `bin/`) runs the script in a subshell — not what we want.

Onward:

```
deactivate () {
...
}
```

This is just a bash function definition. Function calls work just like commands in bash. Now we know that the commands included in this block are what runs when we say deactivate in our virtual environment.

The meat of the `activate` file is in lines 42 – 47:

```
# unset irrelevant variables
deactivate nondestructive

VIRTUAL_ENV="/Users/afk/examples/testenv"
export VIRTUAL_ENV

_OLD_VIRTUAL_PATH="$PATH"
PATH="$VIRTUAL_ENV/bin:$PATH"
export PATH
```

Starting with a call to `deactivate` ensures that any existing virtual environment is deactivated before a new one is created. Virtual environments are separate from each other; they can't be nested.

The rest of this is pretty straightforward. `export` is the only other bash command we need to know, and it's really simple: it just exports a variable into your current environment. It also ensures that environmental variables in processes spawned from the current one get the same values. Since we're running the file via `source`, the effect is to set variables and then keep them after the activate script finishes running.

So the activate script does three primary things:

1. Sets a VIRTUAL_ENV bash environmental variable containing the virtual environment directory
2. Prepends that directory to your PATH
3. Sets the new PATH.

What is PATH? The PATH is an environmental variable representing a list of directories. Your system will look for programs and scripts in the order that directories are listed. The list is separated by colons.

Let's see this in action. To see what your PATH looks like before you run the activate file, hop into your terminal and type echo $PATH. This prints out the value of PATH to the terminal. Mine looks in part like this (I've inserted line breaks for clarity):

```
/usr/local/bin:
/usr/local/sbin:
/usr/bin
```

All this says is that when I type a command like `python`, my system looks first in `/usr/local/bin` for python. If it can't find it, it moves on to `/usr/local/sbin`, then to `/usr/bin`, and so on.

Now let's run the `activate` file and see what changed. (Again, I've inserted line breaks for clarity.)

```
testenv\ $ source bin/activate
(testenv)testenv\ $ echo $PATH
/Users/afk/examples/testenv/bin:
/usr/local/bin:
/usr/local/sbin:
/usr/bin
```

Sure enough, that `testenv` directory has been prepended to my PATH. Now bash will look for python, or any other system command, first in the `bin/` directory here in my `testenv`. What's in there? Let's take a look:

```
testenv\ $ ls bin/
activate            easy_install        python
activate.csh        easy_install-2.7    python2
activate.fish       pip                 python2.7
activate_this.py    pip-2.7
```

There's our `activate` file that we've been examining, plus a version of python! So this is the python installation that will get modified if we install packages, and the python that will be run by `python`. For easy confirmation of this, we can use `which`:

```
(testenv)testenv\ $ which python
/Users/afk/examples/testenv/bin/python
```

We're using the `testenv` python, not the `system` python (which is found in `usr/bin/`).

Notice that `activate` also modified my bash prompt (PS1). We'll skip some of the details here — the important point is that this code stores your old `PS1` and inserts the name of the virtualenv into the new one.

We're done with our virtualenv for now — let's come back to `deactivate`.

```
deactivate () {
    unset pydoc

    # reset old environment variables
    if [ -n "$_OLD_VIRTUAL_PATH" ] ; then
        PATH="$_OLD_VIRTUAL_PATH"
        export PATH
        unset _OLD_VIRTUAL_PATH
    fi
    if [ -n "$_OLD_VIRTUAL_PYTHONHOME" ] ; then
        PYTHONHOME="$_OLD_VIRTUAL_PYTHONHOME"
        export PYTHONHOME
        unset _OLD_VIRTUAL_PYTHONHOME
    fi

    #[special case omitted for brevity]

    if [ -n "$_OLD_VIRTUAL_PS1" ] ; then
        PS1="$_OLD_VIRTUAL_PS1"
        export PS1
        unset _OLD_VIRTUAL_PS1
    fi

    unset VIRTUAL_ENV
    if [ ! "$1" = "nondestructive" ] ; then
    # Self destruct!
        unset -f deactivate
    fi
}
```

`deactivate` calls `export` to restore the old environmental variables, then calls `unset` to remove unneeded variables from the environment. (You can verify this from the terminal by using the command `env` to view all your environmental variables.) Finally, `deactivate` calls `unset -f deactivate` to remove the `deactivate` function itself. (`-f` removes a function.) The function is now gone from the environment, which you can easily verify:

```
(testenv)testenv\ $ deactivate
testenv\ $ deactivate
-bash: deactivate: command not found
```

Our PS1, PATH, and PYTHONHOME end up with their original values.

There you have it — no magic, and just a tiny bit of bash scripting to understand the power of a virtual environment. ■

Allison is a facilitator at Hacker School, a writer's retreat for experienced programmers in New York City. She studied astrophysics and dabbled in finance before diving in to programming at Hacker School.

Reprinted with permission of the original author. First appeared in *hn.my/virtualenv* (hackerschool.com)

stripe

Accept payments online.

Go & Assembly

By CALEB DOXSEY

One of my favorite parts about Go is its unwavering focus on utility. Sometimes we place so much emphasis on language design that we forget all the other things programming involves. For example:

- Go's compiler is fast.
- Go comes with a robust standard library.
- Go works on a multitude of platforms.
- Go comes with a complete set of documentation available from the command line/a local web server/the internet.
- All Go code is statically compiled so deployment is trivial.
- The entirety of the Go source code is available for perusal in an easy online format.
- Go has a well-defined (and documented) grammar for parsing (unlike C++ or Ruby).
- Go comes with a package management tool. go get X (for example, `go get code.google.com/p/go.net/websocket`).
- Like all languages Go has a set of style guidelines, some enforced by the compiler (like uppercase vs. lowercase) and others that are merely conventional, but it also has a tool to clean up code: `gofmt name_of_file.go`
- And there's also go fix, which can automatically convert Go code designed for earlier versions to newer versions.
- Go comes with a tool to test packages: `go test /path/to/package`. It can do benchmarks too.
- Go is debuggable and can be profiled.
- Did you know there's a playground [play.golang.org] to try Go online?
- Go can interact with C libraries via cgo.

Those are just a few examples, but I want to focus on one that's not generally well known: **Go can seamlessly use functions written in Assembly**.

How to Use Assembly in Go

Suppose we want to write an assembly version of a `sum` function. First, create a file called `sum.go` that contains this:

```
package sum

func Sum(xs []int64) int64 {
  var n int64
  for _, v := range xs {
    n += v
  }
  return n
}
```

This function just adds a slice of integers and gives you the result. To test it, create a file called `sum_test.go` that contains this:

```
package sum

import (
  "testing"
)

type (
  testCase struct {
    n int64
    xs []int64
  }
)

var (
  cases = []testCase{
    { 0, []int64{} },
    { 15, []int64{1,2,3,4,5} },
  }
)

func TestSum(t *testing.T) {
  for _, tc := range cases {
    n := Sum(tc.xs)
    if tc.n != n {
      t.Error("Expected", tc.n, "got", n, "for",
tc.xs)
    }
  }
}
```

Writing tests for your code is generally a good idea, but it turns out that for library code (anything not `package main`) it also makes for a good way to experiment. Just type `go test` from the command line and it will run your tests.

Now let's replace this function with one written in assembly. We can start by examining what the Go compiler produces. Instead of `go test` or `go build`, run this command: `go tool 6g -S sum.go` (for a 64-bit binary). You should see something like this:

```
--- prog list "Sum" ---
0000 (sum.go:3) TEXT    Sum+0(SB),$16-24
0001 (sum.go:4) MOVQ    $0,SI
0002 (sum.go:5) MOVQ    xs+0(FP),BX
0003 (sum.go:5) MOVQ    BX,autotmp_0000+-16(SP)
0004 (sum.go:5) MOVL    xs+8(FP),BX
0005 (sum.go:5) MOVL    BX,autotmp_0000+-8(SP)
0006 (sum.go:5) MOVL    xs+12(FP),BX
0007 (sum.go:5) MOVL    BX,autotmp_0000+-4(SP)
0008 (sum.go:5) MOVL    $0,AX
0009 (sum.go:5) MOVL    autotmp_0000+-8(SP),DI
0010 (sum.go:5) LEAQ    autotmp_0000+-16(SP),BX
0011 (sum.go:5) MOVQ    (BX),CX
0012 (sum.go:5) JMP     ,14
0013 (sum.go:5) INCL    ,AX
0014 (sum.go:5) CMPL    AX,DI
0015 (sum.go:5) JGE     ,20
0016 (sum.go:5) MOVQ    (CX),BP
0017 (sum.go:5) ADDQ    $8,CX
0018 (sum.go:6) ADDQ    BP,SI
0019 (sum.go:5) JMP     ,13
0020 (sum.go:8) MOVQ    SI,.noname+16(FP)
0021 (sum.go:8) RET     ,
sum.go:3: Sum xs does not escape
```

Assembly can be quite difficult to understand and we will take a look at this in more detail in a bit, but first let's go ahead and use this as a template. Create a new file called `sum_amd64.s` in the same folder as `sum.go`, which contains this:

```
// func Sum(xs []int64) int64
TEXT ·Sum(SB),$0
    MOVQ    $0,SI
    MOVQ    xs+0(FP),BX
    MOVQ    BX,autotmp_0000+-16(SP)
    MOVL    xs+8(FP),BX
    MOVL    BX,autotmp_0000+-8(SP)
    MOVL    xs+12(FP),BX
    MOVL    BX,autotmp_0000+-4(SP)
    MOVL    $0,AX
    MOVL    autotmp_0000+-8(SP),DI
    LEAQ    autotmp_0000+-16(SP),BX
    MOVQ    (BX),CX
    JMP     L2
L1: INCL    AX
L2: CMPL    AX,DI
    JGE     L3
    MOVQ    (CX),BP
    ADDQ    $8,CX
    ADDQ    BP,SI
    JMP     L1
L3: MOVQ    SI,.noname+16(FP)
    RET
```

Basically all I did was replace the hardcoded line numbers for jumps (JMP, JGE) with labels and added a middle dot (·) before the function name. (Make sure to save the file as UTF-8.) Next, we remove our function definition from our `sum.go` file:

```
package sum

func Sum(xs []int64) int64
```

Now you should be able to run the tests again with `go test`, and our custom assembly version of the function will be used.

How it Works

This type of assembly is described in more detail here [hn.my/asm]. I will briefly explain what it's doing.

```
MOVQ    $0,SI
```

First, we put 0 in the SI register, which is used to represent our running total. The Q means quadword, which is 8 bytes, and later we'll see L, which is for 4 bytes. The parameters are in (source, destination) order.

```
MOVQ    xs+0(FP),BX
MOVQ    BX,autotmp_0000+-16(SP)
MOVL    xs+8(FP),BX
MOVL    BX,autotmp_0000+-8(SP)
MOVL    xs+12(FP),BX
MOVL    BX,autotmp_0000+-4(SP)
```

Next, we take the parameter passed in and store its value on the stack. A Go slice is made up of 3 parts: a pointer to its location in memory, a length and a capacity. The pointer is 8 bytes while the length and capacity are 4 bytes each. So this code copies those values through the BX register.

```
MOVL    $0,AX
MOVL    autotmp_0000+-8(SP),DI
LEAQ    autotmp_0000+-16(SP),BX
MOVQ    (BX),CX
```

Next, we put 0 in AX, which we'll use as an iterator variable. We load the length of the slice into DI and load xs' elements pointer into CX.

```
    JMP     L2
L1: INCL    AX
L2: CMPL    AX,DI
    JGE     L3
```

Now we get to the meat of the code. First, we jump down to L2 where we compare AX and DI. If they're equal, we've consumed all the items in the slice so we go to L3. Basically `i == len(xs)`.

```
MOVQ    (CX),BP
ADDQ    $8,CX
ADDQ    BP,SI
JMP     L1
```

This does the actual addition. First, we get the value of CX and store it in BP. Then, we move CX 8 bytes ahead. Finally, we add BP to SI and jump to L1. L1 increments AX and starts the loop again.

```
L3: MOVQ    SI,.noname+16(FP)
  RET
```

After we've completed our summation, we store the result after all the arguments to the function (so 16 bytes ahead because a slice is 16 bytes). Then we return.

Rewritten

Here's my rewrite of the code:

```
// func Sum(xs []int64) int64
TEXT ·Sum2(SB),7,$0
    MOVQ    $0, SI       // n
    MOVQ    xs+0(FP), BX // BX = &xs[0]
    MOVL    xs+8(FP), CX // len(xs)
    MOVLQSX CX, CX       // len as int64
    INCQ    CX           // CX++

start:
    DECQ    CX           // CX--
    JZ done              // jump if CX = 0
    ADDQ    (BX), SI     // n += *BX
    ADDQ    $8, BX       // BX += 8
    JMP start

done:
    MOVQ    SI, .noname+16(FP) // return n
    RET
```

Hopefully it's a little easier to understand.

Caveats

It's pretty cool that you can do this, but it's not without its caveats:

- Assembly is hard to write and especially hard to write well. Compilers will often write faster code than you will (and going forward the Go compiler will get better at this).
- Assembly only runs on one platform. In this case, the code I wrote will only work on amd64. One solution to this problem is to supply a Go version of the code and call it from x86 and arm.
- Assembly ties you down in ways standard Go code won't. For example, the length of slice is currently a 32-bit int, but it won't be for long. This code will break when that change is made (and it will break in nasty ways the compiler can't detect).
- Currently the Go compiler will not inline functions written in Asembly, but it will inline small Go functions. So counterintuitively you can actually make your program slower by using Assembly.

Still it's useful for at least two reasons:

1. Sometimes you need the power Assembly can give you, either for performance reasons or the need for some highly specialized operation available in the CPU. The Go source code includes several good examples of when it's appropriate to use.
2. This is actually a great way to learn Assembly, since it's very easy to get started. ■

Caleb is a Software Developer at SendHub and the author of the book An Introduction to Programming in Go [golang-book.com]. He blogs haphazardly at *doxsey.net*

Reprinted with permission of the original author.
First appeared in *hn.my/goassembly* (doxsey.net)

How A Pull Request Rocked My World

By CLAY ALLSOPP

This is a story about Pull Request #2 and how it made me rethink my code. This isn't really a story about the code itself; it's about how even a small refactoring can multiply value and productivity. But hey, let's not get ahead of ourselves.

Long ago, there was some code in an iOS library called Formotion [hn.my/formotion] that looked like this:

```
if row.submit_button?
    make_submit_cell(row, cell)
elsif row.switchable?
    make_switch_cell(row, cell)
elsif row.checkable?
    make_check_cell(row, cell)
elsif row.editable?
    make_text_field(row, cell)
end
```

Basically, it configures this `cell` object based on properties of the `row`. That serpentine `if/elsif` chain did the trick and didn't seem alarmingly unreasonable at the time, so the code shipped with the first version of Formotion. It all worked out, the project got some watchers and star-ers, and folks generally had great feedback. All was well.

But then a little pull request [hn.my/pull2] appears a few days later.

I get an email about it, and am immediately excited because, hey, someone liked my little project enough to contribute! So I command-click the link open in a new tab and have a look.

The request is labeled **Refactoring & Multiline Text Type**. So usually when someone comes up to you and says like, "Hey, I did some refactoring, check it out," the implicit understanding is that the original thing just wasn't so hot. And about nine times out of ten you agree and know that it's true, but it is a little awkward when someone beats you to fixing it.

Anyway, I inhale whatever pride I have about my work and click the "Files Changed" tab to investigate. I take a gander, and basically what the request does is replace that big fat `if/elsif` block with this one line:

```
row.object.build_cell(cell)
```

Whoa, what is this new row.object? Well, elsewhere in the refactor, this line appeared:

```
@object = Formotion::RowType.for(type).new(self)
```

What's all this about? Turns out that the refactor took the various row configurations (`switch`, `submit`, `check`) and created new subclasses of `Formotion::RowType` for `each:SwitchRow`, `SubmitRow`, `CheckRow`, etc. In those objects, the `build_cell` method now dictates how the `cell` is configured, taking the place of the old `make_submit_cell` methods.

And that `RowType.for()` method? It actually does a little metaprogramming to grab the appropriate subclass:

> **“Reorganizing your code (in the right way) can create cascading growth in productivity and value.”**

```
# type == :switch or :submit etc
string = "#{type.to_s.downcase}_row".camelize
# string == 'SwitchRow', 'SubmitRow' etc
return Formotion::RowType.const_get(string)
```

So instead of some `if` tree and hard-coded methods, everything is dynamic and decided at run-time, plus you get a great plug-in architecture if you do some polymorphic tricks with these RowTypeobjects. And all of these benefits came from just one simple refactoring. The power of small refactors is absolutely crazy. Brilliant.

All of this blew me away. Like this “@mordaroso” fellow strolled on up, cocked his knee skyward, and jettisoned his leg into the rickety wooden door that previously sheltered my mind.

So there I am, hunched over my laptop, eyebrows furrowed to the point where my forehead folds in over itself, just stupefied by this code. And the ramifications, like how it allows for extensibility and plugins and testing and a generally more robust codebase, are just gushing into the void Fabio Kuhn exposed in said cranium.

With just that one refactor, the value of my little library had grown tremendously! If you wanted to make a plug-in, there was no messy monkey-patching: simply create your new `RowType` subclass, like `MyWidgetRow`, and you’re done. It just works.

Not a day goes by where I’m shin-deep in a project and don’t think about what that pull request revealed: reorganizing your code (in the right way) can create cascading growth in productivity and value. In Formotion’s case, that pattern of dynamic-class-lookup is now used in other parts of the codebase and allows for a project-wide plug-in architecture. Big win all around. This sort of stuff really is worth being diligent about.

So thanks, Fabio, for blowing my mind. ■

Clay Allsopp is a hacker, Thiel Fellow, and internet enthusiast. An iOS developer since day one, Clay has crafted beautiful mobile apps with over a million cumulative downloads for startups like Circle. He is currently building Propeller [usepropeller.com], the best way for anyone to build a mobile app.

Reprinted with permission of the original author.
First appeared in *hn.my/pull* (clayallsopp.com)

How To Open A Microchip And What's Inside

By SVARICHEVSKY MIKHAIL

Microchips can indeed be considered a black box: as long as it's working you normally don't look inside.

But what if you want to?

Today we'll demonstrate how to "open" chips and show you what's inside.

WARNING! All operations with concentrated (and especially hot) acids are extremely dangerous. Only trained persons should work with them using required protective equipment (acid-proof gloves, protection glasses, protective suit, fume hood and more). Remember that you only have 2 eyes!

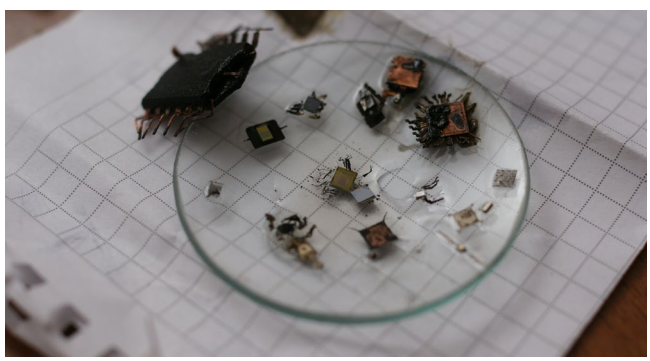
This article is for educational purposes only, do not try to repeat!

Opening Microchips

Take some microchips of interest and add concentrated sulfuric acid. Container should be closed, but not airtight, so that fumes can escape (that is extremely important). Heat it to boiling point (300 °C). The white substance at the bottom is baking soda; it's here to neutralize accidental spills and some of the fumes.

After 30-40 minutes, acid "burns" plastic to carbon:

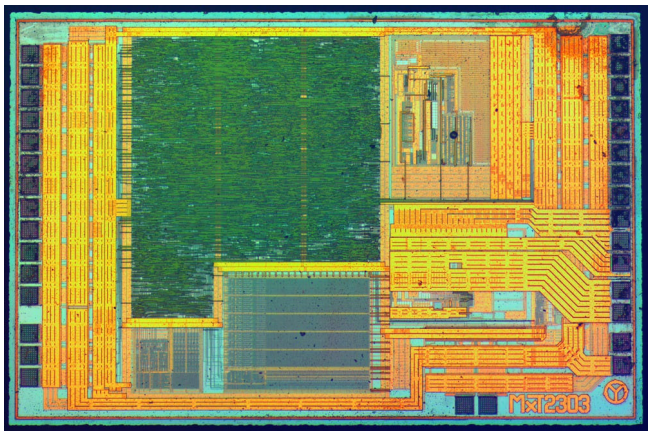
After it cools down, we can sort what is ready for the next step and what needs another acid bath (thick, bulky packages usually need 2-3 rounds):



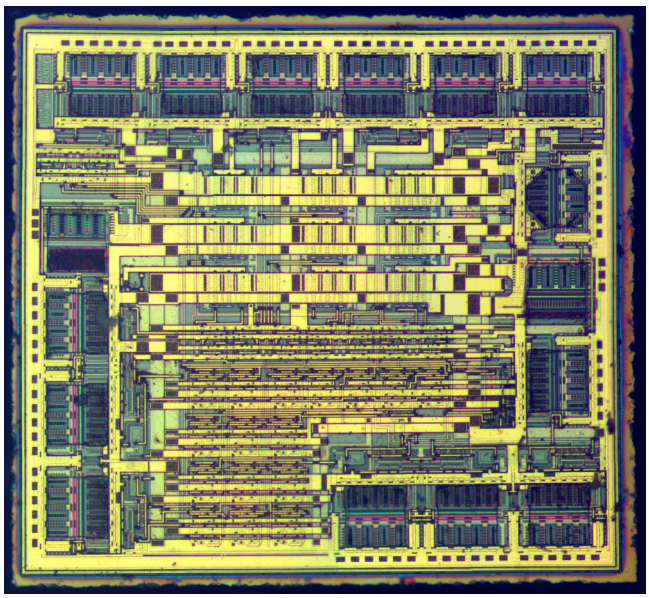
If pieces of carbon stick to the microchip itself and cannot be removed mechanically, one can remove them in hot concentrated nitric acid (temperature is much lower, ~110-120 °C):

Taking a Look

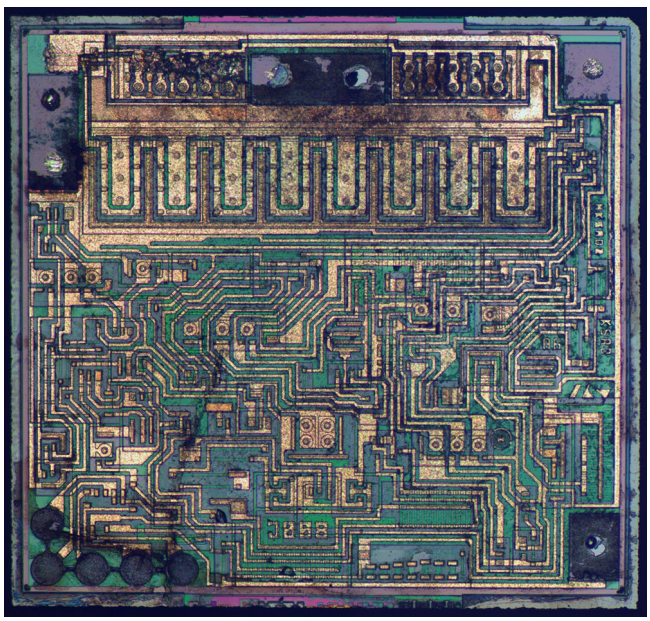
Colors are enhanced; in reality they are much less saturated.



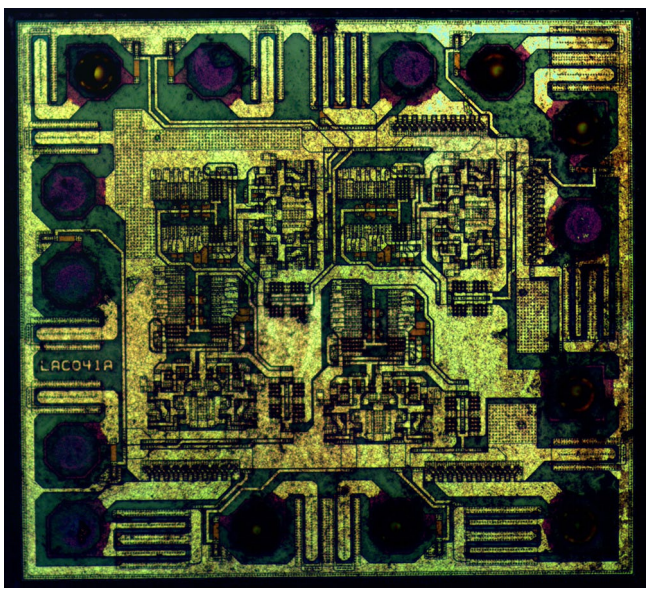
PL2303HX — USB<>RS232 converter, chips like this are used in Arduino-like boards for example.



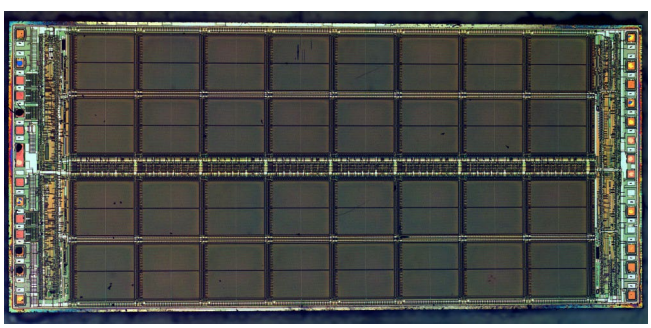
74HC595 — 8-bit shift register.



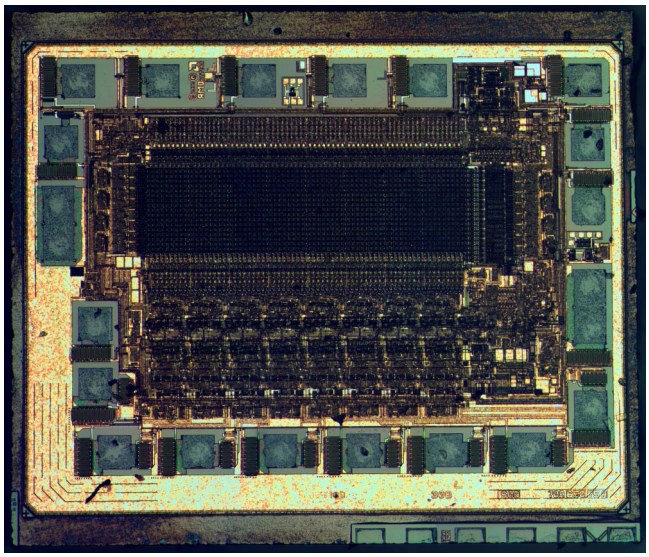
LM1117 — low-dropout linear regulator.



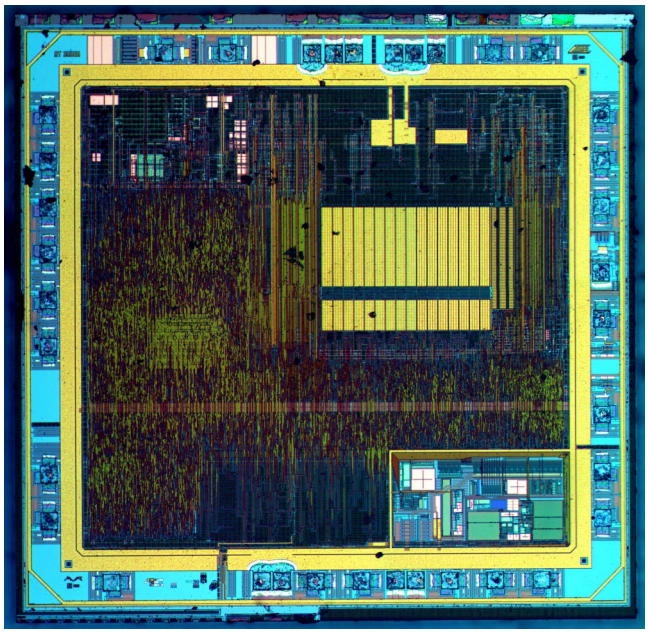
NXP 74AHC00 — quad 2-input NAND gate.
This is a nice example that "old" tech nodes (1µm and older) are still in use.
Also, note how many spare via are there for improved yield...



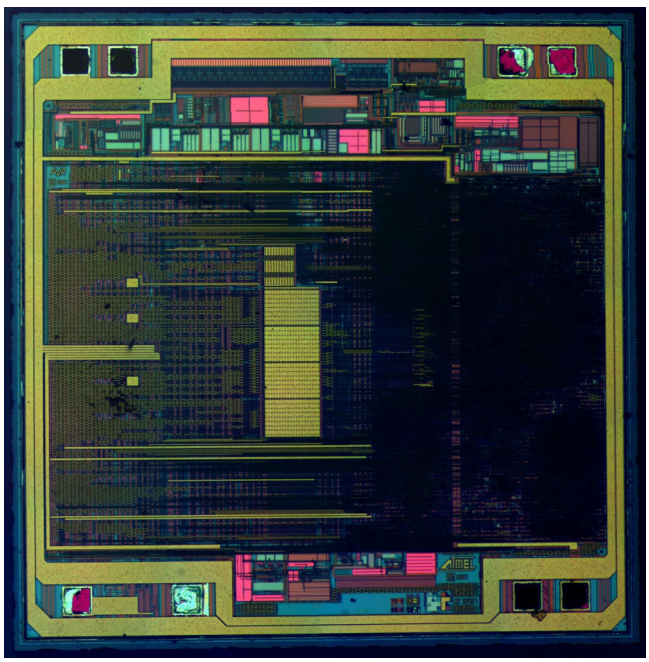
Micron MT4C1024 — 1 mebibit (220 bit) dynamic ram.
Widely used in 286- and 386-era computers in the early '90s.



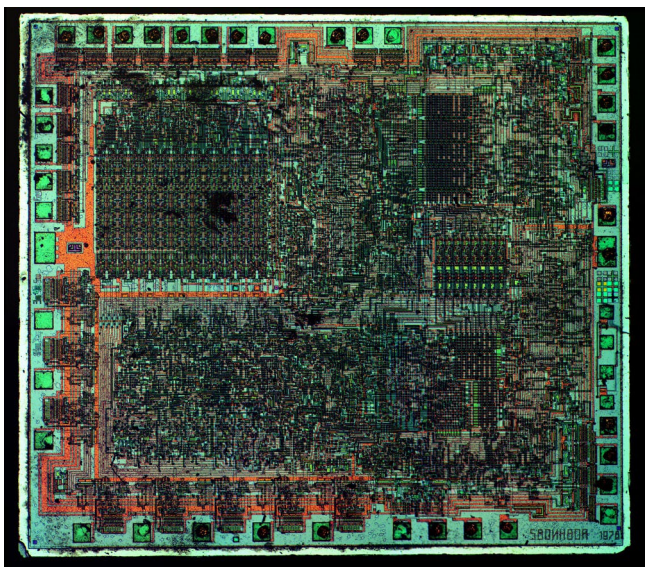
AMD Palce16V8h GAL is an 32x64 array of AND elements.
GAL(Generic array logic) microchips are FPGA and CPLD grandfathers.



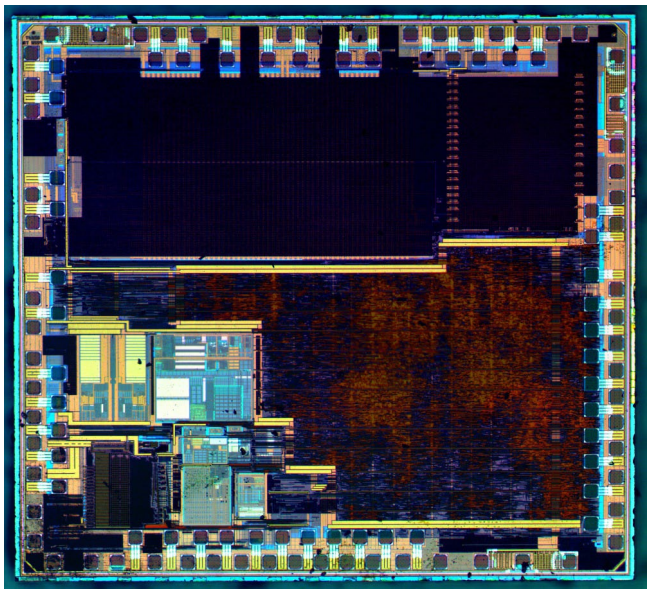
ATmega8 — one of the most popular 8-bit microcontrollers.



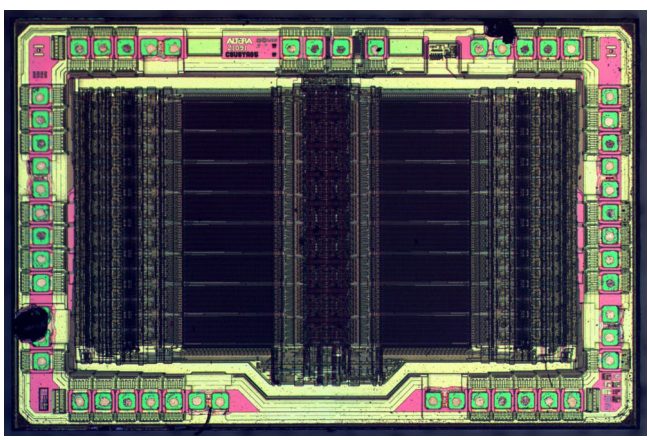
ATtiny13A — one of the smallest Atmel's microcontrollers: only 1kb of flash and 32 bytes of SRAM.



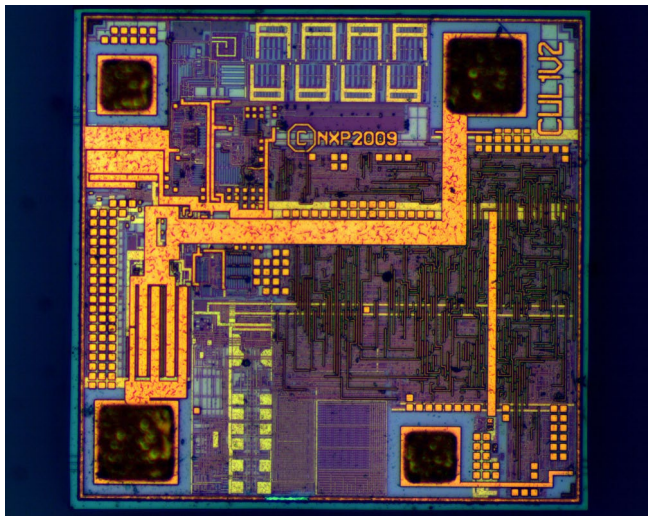
KR580IK80A (later renamed to KR580VM80A) - one of the most widespread Soviet processors. Contrary to popular belief, it appeared to be not an Intel 8080A (or 8080) clone, but a code-compatible redesign (while several parts are quite similar, routing is different as well as pad placement).



STM32F100C4T6B — is the smallest microcontroller made by STMicroelectronics based on ARM Cortex-M3 core.



Altera EPM7032 — Altera EPM7032 - CPLD that have seen a lot... One of the last using 5V supply.



MIFARE chip, used in Moscow's subway RFID tickets.

Mikhail is a self-taught electronics engineer. He left his IT job years ago to start fabless startup Zeptobars — hopefully will be making microcontrollers.

Reprinted with permission of the original author.
First appeared in *hn.my/microchip* (zeptobars.com)

Using Parkinson's Law to Kick Procrastination's Ass

By SEAN GRANSEE

I've recently made four lifestyle changes that have allowed me to get more done and put much more effort into everything I do, all while feeling great with very little stress.

1. I sleep 8 hours a day.
2. I work out for an hour every weekday.
3. I hide all clocks while I'm working.
4. I don't do anything related to academics on Saturdays and past 5pm on weekdays.

My main focus of this post is the last two, but I'll briefly address the first two because I think they're very important.

Sleep

I read a lot of what tech entrepreneurs have to say, and I've noticed a trend. The ones who are most successful seem to be the ones who value their physical and mental health. I feel like every few months I see a new study about how getting adequate sleep makes you perform better. It's almost common knowledge at this point, yet so few students take that research to heart.

Working more hours in a day doesn't necessarily correlate with getting more done. I find that I'm much more efficient and productive if I've slept well the night before. Therefore, I've been putting myself on a consistent sleep schedule from 1am – 9am every weeknight. After a few days, I started being able to fall asleep much faster and wake up refreshed in the morning… something I haven't consistently felt in months.

Exercise

I also made a habit of going to the gym every day from 5 – 6pm. In the beginning this was a pain, and I felt like it was taking an hour away from me getting work done. I used to feel like I didn't have time to exercise, but now I feel like I don't have time NOT to exercise. By forming a habit, it feels like something I need to do every day, just like eating and showering. I don't think twice about it anymore.

Parkinson's Law

But my real trick for getting more done with much less stress is in the things that I don't do. It all stems directly from Parkinson's Law:

> *Work expands so as to fill the time available for its completion.*
> *— Cyril Northcote Parkinson*

The idea has been around since 1955, and taking advantage of it has completely changed the way I work. Parkinson's Law, in other words, states that if you have a certain amount of time to complete something, that's how long it will generally take.

Usually this means that if you have more time than you need to get something done, you'll use that extra time because it's available to you. From personal experience I've found the reverse to be true as well. If I give myself less time to do something, I'll still manage to get it done in that limited time.

> **“Time usually isn’t the bottleneck when it comes to getting things done. The real bottleneck is focus.”**

Focus

Scott H. Young wrote in his email newsletter that time usually isn’t the bottleneck when it comes to getting things done. The real bottleneck is focus. I found that to be absolutely true when it came to my study habits. Everything I did took 3-4 times longer than it should because I would distract myself every few minutes by Facebook, email, text messages, Hacker News, or any number of other distractions.

But the real time costs when it comes to these distractions aren’t the distractions themselves. It’s the context switching that occurs every time I’m distracted. Thirty seconds on Facebook may seem harmless, but the real time cost comes from the time it takes me to become mentally re-engaged with a task after a distraction. Read up on context switching, and the findings may surprise you.

Everything up until this point led me to believe that I would be much more productive if I eliminated distractions. If I could frequently achieve flow (being fully immersed in a task), then I’d be able to get things done more than twice as fast, right? So now comes the mission of eliminating distractions.

Distractions

The biggest distraction in my life (and potentially yours) isn’t Facebook, email, friends, or anything remotely related. It’s the clock. Whenever a clock is in my field of vision, I find myself constantly looking at it. Every time I look at the clock, some sort of thought about how much time I have left enters my head. Every time that happens, I waste time because of context switching.

Cover the clocks

So I eliminate clocks whenever I’m working. I disabled the clock in the corner of my laptop. Since you can’t disable the clock on an iPad, I ripped the sticky part off a post-it note and covered the clock. My phone isn’t a problem because I usually have that on “do not disturb” mode when I’m working.

The only genuinely useful thing about clocks when I’m working is that they tell me when I have to be somewhere. But I don’t need a clock for that. All I need is an alarm. If I don’t have to be somewhere until noon, I’m much better off not knowing what time it is until it’s 11:45. I’m able to get by without ever looking at the clock because I set an alarm before I have to be anywhere. I use an app called Brrrr Alarm [hn.my/alarm], which vibrates your phone instead of making noise, all the time when I’m working in a quiet space.

Let’s think back to Parkinson’s Law: “Work expands so as to fill the time available for its completion.” The no-clocks thing works great for short tasks. When I don’t know how much time I have to complete a task, I naturally work as quickly as possible the entire time.

But aside from eliminating some context switching, this alone doesn’t really do much to help me efficiently complete long-term tasks. What harm can a little Reddit break do when I’m working on something that’s due next week?

No studying after 5pm

The answer is simple. It might sound crazy, but rigorously following this rule has forced me to completely focus most of the time and start things long before they're due. I don't do anything related to academics on Saturdays and past 5pm on weekdays.

I like to think of this as reverse-procrastination. Think of how procrastination works. You put a task off as long as possible, and then complete it hours before it's due. In a way, you're utilizing Parkinson's Law when you procrastinate. You're limiting the time you have to do something, and therefore you're able to complete it in much less time than it would normally take you. In a way, procrastinators are very efficient when it comes to how long it takes them to get things done. But this comes at a great cost: stress.

> *If I give you a week to complete [a short] task, it's six days of making a mountain out of a molehill. If I give you two months, God forbid, it becomes a mental monster. The end product of the shorter deadline is almost inevitably of equal or higher quality due to greater focus.*
> *— Timothy Ferriss*

It's stressful to do everything at the last minute. So I've taken procrastination and turned it on its head. I tell myself that the due date for everything I need to do each day is at 5pm. This forces me into consistent periods of ultra-efficient productivity because I'm constantly racing against time to get everything done by 5pm. I find myself not even wanting to take those quick breaks because they might stop me from completing everything I need to do by 5pm.

This constraint I've imposed on myself has also caused me to start tasks long before I need to. Before I started imposing this constraint, I generally wouldn't start anything until the day before it was due. Now, if I've finished everything I need to get done for tomorrow, I'll go ahead and start something that's not due until next week. Why? Because I don't want to get stuck in a situation a week from now where that task isn't done by 5pm. I now spend almost every moment before 5pm in my day actually working.

Here's what a typical week looks for me now:

Weekdays

10am – 5pm: Homework, studying, going to class.
5pm – 6pm: Gym.
6pm – 1am: Whatever I want, as long as it has nothing to do with academics.
1am – 9am: Sleep.

Saturdays:

Whatever I want, as long as it has nothing to do with academics.

Sundays

No constraints. I can use this as a catch-up day if I absolutely need it.

I'll occasionally make exceptions to those constraints for group work or scheduled school events like review sessions and exams. Other than that, I stick to this schedule pretty rigorously.

By constraining the amount of time I have to study and do homework, I force myself to completely focus and put more effort into my studies with much less stress. This, combined with getting adequate sleep and exercise, has made me much happier and more efficient these past few weeks. It may or may not work for you, but this combination of lifestyle choices has been tremendously beneficial for me. ■

Sean is an alumnus of the hackNY summer fellowship program and a total sucker for startup culture. He didn't always plan on being a hacker, but the appeal of being able to quickly implement new ideas was simply too much to avoid. He currently resides in Chicago.

Reprinted with permission of the original author.
First appeared in *hn.my/parkinsons* (seangransee.com)

```
{
    join: 'Intensive Online Bootcamp',
    learn: 'Web Development',
    goto: 'http://www.gotealeaf.com'
}
```

Tealeaf Academy
an online school for developers

Learn Ruby on Rails | Level up Skills | Launch Products | Get a Job

MEMSET® HOSTING

Rent your IT infrastructure from Memset and discover the incredible benefits of cloud computing.

MINISERVER™ CLOUD COMPUTE

From $0.020/hour
to 4 x 2.9 GHz Xeon cores
31 GBytes RAM
2.5TB RAID(1) disk

MEMSTORE™ CLOUD STORAGE

$0.091/GByte/month or less
99.999999% object durability
99.995% availability guarantee
RESTful API, FTP/SFTP and CDN Service

SCAN THE CODE FOR MORE INFORMATION

MEMSET® HOSTING

Find out more about us at
www.memset.com

or chat to our sales team on
0800 634 9270.

CarbonNeutral® hosting

THE BRITISH ASSESSMENT BUREAU ISO9001

BSI™ UKAS INFORMATION SECURITY MANAGEMENT 003

THE BRITISH ASSESSMENT BUREAU ISO14001