How to admin

How to assess what's happening within a server more quickly?

- Make sure that you verify the client first: How to verify clients
- Are you being called? Have you never done a call before? Check out Phones!
- Check email and chat to ensure that the issue isn't something that has an obvious reason (maybe some testing or network maintenance is happening).
- Are you on a server node? All the information you need is here: Tracking/Correcting Hardware Node Load Issues
- Check the disk space.

```
df -h; df -ih
```

If you see a disk space problem you can run this to get a nice output of the disk space usage that's there:

```
du -h / | grep ^[0-9.]*G
```

If you see an inode problem (rare, but it does happen!) you can run something like this to get inode usage per folder:

```
echo "Inode usage for: $(pwd)" ; for d in `find -maxdepth 1 -type d |cut -d\/ -f2
|grep -xv . |sort`; do c=$(find $d |wc -l) ; printf "$c\t\t- $d\n" ; done ;
printf "Total: \t\t$(find $(pwd) | wc -l)\n"
```

- Run top. Look at threads. Usually you want to sort by memory usage and not cpu - you can switch what you're sorting by holding shift and then pressing < and > on your keyboard respectively to go left or right. One to the right will sort by memory (it sorts by CPU by default). If you are having issues catching processes in action you can always run ps, as seen further down in this guide.
- If you see an issue with IO please type in iotop and see if it works. If it does, it's top, with IO. You should be able to see the offending process. If it doesn't, run iostat -x: this will show disk access per partition, and can help.
- free -m will show you quick RAM usage. Make sure you understand how RAM works on Linux (http://www.linuxatemyram.com/). Virtuozzo runs swap in RAM itself so ignore if swap is maxed out on VPSs.
- If it is MySQL, check the MySQL bit further down to see if you can either speed it up, repair the tables, or see what the issue is. If it's exim, please go to exim in this guide.

- If they're having weird problems specifically with connecting to their sites, check to see if synflood or lf_bind are on in their CSF configuration! (Check CSF/cpHulkd further down if you're unsure.)
- If it's a process, press c to see what the full command line is that it's running! If that doesn't help, please use strace and lsof to determine what the process is doing (as written in this guide).
- There is a list of logfiles further down in this as well that may be useful if you're having issues with a specific process not working, or if you need logs to determine exactly what's happening. If you are out of other options, ask Google! Please note that Google and experience was used to make the vast majority of this guide! You can also use the 'man' command with any commands on your server to discover new things about it.
- If you are still having problems, please feel free to ask another admin! (Then have them tell me what it was, so I can add it to the guide here.)
- If you need to log processes after this to catch issues in the future, please use either sys-snap or How to use ATOP .
- The rest of this guide has various useful things and helpful tips in general for our environment, as well as links to more in-depth walkthroughs.

Log locations and user info files

- Apache 403 or 500 error? Run this:

```
tail -f /usr/local/apache/logs/error_log
```

  And refresh the page. Each time you do, you'll see the exact error message that's causing your error. If this isn't working but you know the exact page you want to check, you can use the domain logs at /usr/local/apache/domlogs/(domain.domain) to check on it.
- MySQL not starting? Run `service mysql restart` and then check this log file:

```
tail -n 50 /var/lib/mysql/`hostname`.err
```

  Chances are, you'll find the reason it's failing to start.
- Is the client blocked from the server? Check their IP address to see if it's blocked:

```
echo -e "IP?"; read IP; grep $IP /var/log/lfd.log
/usr/local/cpanel/logs/cphulkd.log /etc/csf/csf.* /etc/hosts.allow; grep "deny"
/etc/hosts.allow | grep -v "#"; csf -g $IP
```

  If it is, or if you would like to whitelist them anyways, just run this:

```
echo -e "IP?"; read IP; grep $IP /var/log/lfd.log
/usr/local/cpanel/logs/cphulkd.log /etc/csf/csf.* /etc/hosts.allow; grep "deny"
/etc/hosts.allow | grep -v "#"; /usr/local/cpanel/scripts/cphulkdwhitelist $IP;
csf -a $IP ; csf -tr $IP; whmapi1 flush_cphulk_login_history_for_ips ip=$IP
```

- "Why did my (VPS or Hybrid) server's service fail"? Log into the node and check their container ID against memory failures:

```
grep $CTID /var/log/messages
```

  They might look like one of these error lines:

```
# Allocation error
Oct 27 23:06:56 vz123 kernel: [916614.799622] Fatal resource shortage:
privvmpages, UB $CTID.
# Out Of Memory error
Oct 28 08:48:23 svz010 kernel: [9530769.370393] Out of memory in UB $CTID: OOM
killed process 63280 (mysqld) score 0 vm:3873132kB, rss:567148kB, swap:22392kB
```

Some quick and dirty locations follow:

- /var/log/messages is a good log with a lot of information about most things in the system.
- /var/log/lfd.log and /usr/local/cpanel/logs/cphulkd.log are the two firewall logs within the server.
- /usr/local/cpanel/logs/ is cPanel's general error log folder - the error log there will contain most cPanel errors, and the cpbackup and cpbackuptransporter logs will help with those things respectively. There is also /var/cpanel/updatelogs/ for cPanel updates.
- /etc/trueuserdomains has all domains on the server, and all cPanel users.
- There are other logs and files that are important, but these are the most important. If you see any extras you want to add here, please do so!

sed

- sed 's/(phrase 1)/(phrase2)/g' will replace phrase 1 with phrase 2. sed -i does this inline, within a specific file. So, for example, you can run something like this to change nameservers within someone's nameserver text files:

```
sed -i 's/ns1.oldnameserver.com/ns1.newnameserver.com/g' /var/named/*.db; sed -i
's/ns2.oldnameserver.com/ns2.newnameserver.com/g' /var/named/*.db
```

- That's pretty much all I do with sed - remove things from a string. You can also use replace (man replace if you want) but sed is more flexible. You can also use sed in the middle of another thing which I do a lot of, like this:

```
find /home -type f -name php.ini -print -exec sed -i '/memory_limit.*/d' {} ';'

#or I do something like this if I'm importing mysql backups, you can see it can
be /really/ simple:
for i in $(ls); do notsql=`echo $i | sed 's/".sql"//g'; mysql $notsql < $i; done

#On a really basic level, if you're doing substitution only it's just:
sed 's/(thing you want to replace)/(thing you're replacing the first thing
with)/g'

#But sed is insanely extensible and can be used for a gigantic amount of things!
#these are old admin commands and as such probably aren't helpful as actual
commands.
#But you can see how flexible sed is here, and how you can combine known commands
to make something new :) sed also works great with regex as you can see below!
*/5 * * * * export COLUMNS=512 && /bin/echo -e "\n`date`\n" >> /root/sys-log
2>/dev/null && top -m -b -n 1 | sed '1,7 d' | sort -nrk 10| head -n 25 | sed
's/[[:space:]]\+$//' >> /root/sys-log 2>/dev/null && unset COLUMNS >&/dev/null
grep -lr 'a certain string' /var/spool/exim/input/ | sed -e
's/^.*\/\([a-zA-Z0-9-]*\)-[DH]$/\1/g' | xargs exim -Mrm
```

If you need to know anything else, this guide is fantastic: http://www.grymoire.com/Unix/Sed.html

grep

- In general, using grep goes like this: grep (phrase) (file). So, for example, here are two ways to use grep:

```
grep "hack" /home/username/public_html/thisfileisdefinitelynothacked.php
cat /home/username/public_html/thisfileisdefinitelynothacked.php | grep "hack"
```

- -v finds what doesn't match your phrase. So if the example above was grep -v "hack" /home/username/public_html/thisfileisdefinitelynothacked.php , it would list everything that didn't have hack in it.
- Usually when I'm running grep I use grep -iRl: i for no case, R for recursive, and l for list file. This lets me run large greps on large parts of a server without worrying overmuch about being unable to see which file it is in the resulting stream of information. This is an example of something which may run better with GNU Parallel .
- Google or the grep man page is the best source for any other issues/questions: http://linux.die.net/man/1/grep

awk

- Most of what we're doing is /very/ simple awk stuff. This thing can span entire books very easily. Awk makes it easy to grab something specific from within a string, in a way that cut sometimes doesn't (if you don't know about cut, check out the short intro below). You can for example grab the xth part of a string:

```
echo "this is a string" | awk '{print $3}'
```

  However it's a fully-featured language so it can do basically anything. You can see it here, grabbing any of a bunch of words way faster than grep or other things could:

```
awk 'FNR==1 && /eval/ || /strlen/ || /strto/ || /auth_pass/ || /_dl/ || /GLOBALS/
{ print FILENAME }; FNR>1 {nextfile}'
```

  tldp goes into more detail here: http://tldp.org/LDP/abs/html/awk.html

Regular Expressions (regex)

- These are complex. This will help: http://tldp.org/LDP/Bash-Beginners-Guide/html/chap_04.html
- This is something you just need to read the full guide to grasp. Maybe a few full guides. Test them out on your end. 🙂

cut

- Cut grabs a specific part of a string. The important flags are -d, for 'delimiter', and -f, for 'field'. If you write something like this:

```
cut -d" " -f3
```

  cut will print the third field counting by spaces (the first field is field 1). So if you did something like:

```
echo "my head hurts. why?" | cut -d" " -f3
```

  This would return "hurts." . It is sometimes (usually) cleaner to use awk for this, above, but sometimes cut is faster and simpler (it's a very fast and simple tool) and you don't want to deal with awk.

ps

- Basically all you want:

```
ps -aux
```

http://linux.die.net/man/1/ps man ps to see all the other stuff you can do with it! It's basically just top, but slightly more in-depth, and for a specific moment of time instead of constantly updating (yes you can get top to run once and capture the output, but afaik it isn't as good).

strace

- strace a process will approximately be:

```
strace -p (process ID) -s 200 -tt -o (output file)
```

- run thing and strace will approximately be:

```
strace -s 200 -tt -o (output file) /usr/local/bin/php
/home/cpaneluser/public_html/index.php
```

- For more: A short strace primer

ss/netstat

- All you need to know:

```
netstat -planet
#this outputs the information in a different way but gives you more grouped
information faster. see which you prefer. :) ss is newer and better in general.
ss -planet
```

- There is a more in-depth primer here: http://www.cyberciti.biz/tips/linux-investigate-sockets-network-connections.html
- The man page is also helpful: http://linux.die.net/man/8/ss

lsof

- lsof is great! think of lsof when you want to know where processes are from.

```
lsof -p (process ID) #this will give you all open threads for a process, letting
you track down where it's originating from.
lsof | grep -i DELETED #this will give you all current things that are deleted
but waiting for reboot to "actually" delete files, which can be really helpful
when dealing with OOMs.
```

- If you need anything beyond this there is a short lsof primer here: https://danielmiessler.com/study/lsof/ Again, the man page. Man page for everything. http://linux.die.net/man/8/lsof

Loops and commands

> **PLEASE RUN EVERYTHING WITH ECHO FIRST**
> Please, please please please - run every command you write and are running for the first time with echo first! This is to see what the results are going to be, before you run it (and not after x.x).

For(/while)

- loops are just modal logic. almost everything I write is something I think of in this "put in x, get y" way. so sometimes you'll be like "I want to see the nameservers for every nameserver file on this server", maybe. so you think "well, then:"

```
for (variable) in (nameserver files); do (find nameservers in each file); done
```

  which translates to:

```
for i in $(ls /var/named/*.db); do grep "NS" $i; done
```

  While is a similar sort of command but it's almost never used so it's mostly outside the scope of this. Sometimes, due to the way that cat and for use whitespace, you're going to want to use while read loops that look like this:

```
(command that creates input) | while read ; do line=$REPLY ; (stuff with that
input) ; fi; done;
```

  That leads to writing finished loops that look something like this:

```
grep "wp-login" /usr/local/apache/logs/error_log | while read ; do line=$REPLY ;
if [[ `echo $line | awk '{print $1}'` -gt 200 ]] ; then csf -d `echo $line | awk
'{print $2}' | cut -d: -f2` Blocked for over 200 login attempts to wp-login ; fi;
done;
```

If/Then

- If/Then works like this: if (something happens), then (something results), fi. There is also else: if (thing), then (thing), elif (something else), then (thing), fi. Here is a sample if statement:

```
if [[ grep "pasta" /root/dish.txt ]] ; then echo "Yum, pasta" ; else echo "Aww,
no pasta."; fi;
```

bash specifically

- If you looked at that for loop and thought 'what is the $ for?', or were wondering why nothing above uses !s, this section is for you!

- Bash needs the $ in front of the actual loop if you're running a bash command there, which ... 99% of the time you are. In that 1% you're making a mathematical statement of some sort, generally unneeded in linux work. If you are curious you can look at the guides below.
- Bash also has a few 'special characters'. These are special at all times, and screw you up if you enjoy, for example, putting exclamation points everywhere, or if you really like using the backtick `. A full list is here: http://tldp.org/LDP/abs/html/special-chars.html These are also in regular expressions, generally.
- The most important thing to remember is that \ is 'escape', so it will escape the prior character. So, if you really want that !, you can type in:

```
\!
```

  and it will be processed as though it doesn't break your code.
- This is specifically a link to the 'If' section of the guide, which will help you make if/then statements: http://tldp.org/LDP/Bash-Beginners-Guide/html/sect_07_01.html
- http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO-6.html this and the next chapter are great as well for other non-flag-related items!
- One more thing to remember about bash if you come from a programming background is that variables are untyped.

functions

- In the spirit of this being a very 'these are facts!' document, your function should look like this (sample variables passed in are going to :

```
afunction() {
  var1=$1
  var2=$2
  echo "doing stuff with your first variable $var1"
  echo "doing stuff with your second variable $var2"
  echo "call this by running it like this: afunction something something else"
}
```

- That's all! If you want to know more, you probably shouldn't be writing in bash.

aliases

- Type 'alias' in your command line without the ' marks! Bet you didn't know that those existed, eh? These are shipped in both RedHat and CentOS by default, as well as most other flavors of linux.
- Sometimes in bash you want to run not the alias, for example if ls shows . and .. . There is a flag for that, -A, but it doesn't work out of the box. Good news: you can avoid the alias and properly run the flag! Specifically, you use the backslash and it escapes the alias For example, instead of just running ls, you can run:

```
\ls -A
```

- You can do this for all bash commands that have aliases you despise!

exim

- There used to be a page and should be a page but I don't see it. There are three main commands you want:

```
#grabs all filenames sending mail, good for finding malicious scripts
grep "cwd=" /var/log/exim_mainlog | awk '{for(i=1;i<=10;i++){print $i}}' | sort
|uniq -c| grep cwd | sort -n | grep /home/
#grabs all emails sending mail, good for finding hacked users
grep "A=dovecot_login:" /var/log/exim_mainlog | awk -F 'A=dovecot_login:' '{print
$2}' | awk '{print $1}' | sort | uniq -c | sort -nr
#deletes all bounces, good for cleaning exim queues.
#You can use this to clean full cues by removing the grep command, or can use it
to clean emails from a specific user by changing the grep to something else like
"<user@thing.com".
exim -bp | grep '<>' | awk '/^ *[0-9]+[mhd]/{print "exim -Mrm " $3}' | bash
```

- If you need to harden Exim, you can use this: Exim Hardening Tips

CSF/cpHulk

- There are a few configuration options in CSF/cpHulkd that can greatly slow down a server! They'll be listed in /etc/csf/csf.conf as on if they're an issue - you can just edit that file with your text editor of choice, save, and restart CSF (csf -r) to resolve it.

```
SYNFLOOD #this needs to be turned off
LF_BIND #this needs to be turned off
```

- These configuration options can cause other issues (e.g. mail not sending), they'll have a comment next to the option.

```
SMTP_BLOCK=1 #change this to 0 to turn it off, this blocks lots of email (see CSF
itself for best explanation)
```

- If you're whitelisting someone you can use this one-liner, which asks you for the IP:

```
echo -e "IP?"; read IP; grep $IP /var/log/lfd.log
/usr/local/cpanel/logs/cphulkd.log /etc/csf/csf.* /etc/hosts.allow; grep "deny"
/etc/hosts.allow | grep -v "#"; /usr/local/cpanel/scripts/cphulkdwhitelist $IP;
csf -a $IP ; csf -tr $IP; whmapi1 flush_cphulk_login_history_for_ips ip=$IP
```

- If you're whitelisting multiple people you can just do this after putting them in list.txt:

```
for i in $(cat list.txt); do /usr/local/cpanel/scripts/cphulkdwhitelist $IP; csf
-a $IP ; csf -tr $IP; whmapi1 flush_cphulk_login_history_for_ips ip=$IP; done
```

- This is how you can whitelist all cPanel IPs and also shows how to handle arrays in bash a little bit, if you're curious.

```
declare -a cpanelips=(69.10.42.69 69.175.92.60 208.74.121.106 208.74.121.100
208.74.121.101 208.74.121.102 208.74.121.103 208.74.125.2); for i in $(echo
${cpanelips[@]}); do csf -a $i cPanel Support; /scripts/cphulkdwhitelist $i
cPanel support; done
```

- The logs for these things are all in /var/log/lfd.log and /usr/local/cpanel/logs/cphulkd.log within the server. If you have any issues with them, you'll want to look there to see what the problem is. You can also just turn them off to test if they're actually blocking things, though don't do this for long:

```
csf -x; /scripts/restartsrv_cphulkd --stop #to start: csf -e;
/scripts/restartsrv_cphulkd --start
```

Dealing with malicious files

Clamscan

- Generally you're just going to want to run this:

```
bash <(curl http://files.wiredtree.com/misc/clamscan.sh)
```

- The page, if you want to know more/see more options, is here: Clamscan

Catching Additional Hacked Files

- These files aren't necessarily always malicious so please be careful before you do anything with them (check them first/etc)! Generally you're just going to want to run this:

```
find /home/*/public_html/ -ctime -100 -type f -iname "*.php" -exec awk 'FNR==1 &&
/eval/ || /strlen/ || /strto/ || /auth_pass/ || /_dl/ || /GLOBALS/  { print
FILENAME }; FNR>1 {nextfile}' {} +;
```

- The page, if you want to find anything else, is here: Catch additional hacked files!

find

- this is a good example of most of the find commands you'll want, when you want to use find. Generally it's used to either find something that other things cannot find, or quickly make changes to a large number of files that rm can't handle all at once. This changes directories with 777 permissions to have 755 permissions - you can change the type from d to f for files, can change the -perm to -group or -user to change what it's catching, and can change the chmod to chown or the number to a different one in order to change what it does.

```
find /home/user -type d -perm 0777 -exec chmod 755 '{}' ';'
```

- the other things I use are

```
-mtime # made time
-ctime # creation time
-iname # finds specific name. if you only have part of a name add a wildcard glob
*, e.g.:

find / -maxdepth 1 -type d -iname "roo*" # this will find the 'root' folder

-delete #deletes all files in your find search, this avoids the limitations of
passing them to rm -f so it's way better
-maxdepth #doesn't let your search go past the number of levels put here. e.g.:
-maxdepth 1 will only give you files from the directory you set find to. mindepth
also exists which does the thing in reverse.
-not -path "./directory/*" #ignores anything in your find command output that
would be in the directory you put into the directive.
-print #prints out the results of your command. combine with -depth to run a
delete command without actually deleting the files, -delete assumes -depth ...

#so a good example of all these things (mtime/ctime being similar in use) would
be something like:
find /home/cpuser/public_html/cachedirectory/ -maxdepth 1 -mtime +5 -not -path
"/home/cpuser/public_html/cachedirectory/donotdelete/*" -depth -print
#replace -depth -print with -delete when you are sure it is skipping your
directory :D
```

- A note on time: -mtime +1 means files that are at least two days old. You can use this to extrapolate good methods for finding files from a specific time, e.g. for finding files 'less than 24 hours ago' you have to do -mtime 0. To make this simpler you can use something like -daystart to search from the beginning of today, instead of 24 hours ago. This timing works for all -somethingtime variables.
- Here's a find tutorial from the same place as the sed one, if you have additional questions: http://www.grymoire.com/Unix/Find.html It's a great tool!

MySQL

- Check the mysql logs at /var/lib/mysql/`hostname`.err !
- Also check mysqladmin proc stat. This will show you mysql processes and statistics and will be useful for seeing if there's a single table that's an issue. mysql -e "KILL (ID)" will kill a mysql process.
- Repair tables: mysqlcheck -Ar will repair all tables in all DBs, mysqlcheck -r (db name) will repair all tables in a single DB, mysqlcheck -r (db name) (table name) will repair a single table in a single DB. There is a script to do this for you if you want:

```
bash <(curl http://files.wiredtree.com/misc/databasecheck.sh)
```

- mysql_upgrade --force will sometimes repair mysql issues with the mysql tables. Alternatively you may need to muck about with the mysql version installed within cPanel - that's slightly beyond the scope of this.
- Altering tables to be InnoDB will speed it up if they need to run a ton of queries that don't run fast enough to immediately resolve. You can do this by running: mysql -e "use (database name); alter table (table name) ENGINE=INNODB;"
- Adding an index will speed it up if they are running constant long queries but aren't running multiples.
- You can always run MySQL Tuner if you want to tune someone's MySQL for speed. PLEASE NOTE: this will ALWAYS give you suggestions. You'll get a feeling for where a server generally should be over time but there's no way to get them 'perfect' according to the script.

```
cd /root ; wget -N --no-check-certificate
https://raw.github.com/major/MySQLTuner-perl/master/mysqltuner.pl ; chmod a+x
mysqltuner.pl ; ./mysqltuner.pl
```

- if you've done all this you can delve into Tuning MySQL which goes into this in more detail

Migrations

- The transfer tool is great! cPanel Migration with Root: Transfer Tool
- No root? Officially, cPanel Migration without root: WTmigrate
- Most of my personal opinions on how to do custom migrations are here: Stupid Migration
- This can be helpful if you are doing a custom migration but do not have cPanel access: LFTP
- Have questions about IPPLAN? IP routing (IPPLAN elaboration) can help!

Anything not listed in this guide

- Mostly this is not necessarily rocket science. Look for things first in Confluence, here (there's a search box in the upper right!) by typing in the most specific yet easily-found part of your issue.
- If Google is a better place for it, or you don't know for sure, please do try Google. You can also try stackexchange or any number of sites of that ilk.
- If you think anything else is needed here specifically, talk to me and I can add it