

Unix Quick Reference

This is just a location where I store various commands I found handy for Unix.

Systems Administration

Rapidly wipe multiple hard drives

Nothing beats DBAN [<https://dban.org/>] in ease of use and a feeling of good security. However, I recently had an issue where I had a server with 7 slow hard disks containing data that really wasn't all that sensitive, so I simply wanted to put a bunch of zeros on it, so I booted off of my SystemRescueCD thumbdrive [<https://www.system-rescue.org/>] and ran the following bash script. Should work in any shell which has the *for* command, however.

[wipedrives.sh](#)

```
#!/usr/bin/env bash

# for truly not sensitive information, this command wipes all the OS
information
for drive in a b c d e f g
do
    wipefs -a /dev/sd$drive
done
# but, to really remove in a way that takes tons of effort to recover,
do this also
for drive in a b c
do
    echo Cleaning sd$drive
    dd if=/dev/zero | pv -petrs 580G | dd of=/dev/sd$drive
done
```

I had 7 drives to wipe, and this takes about 5 hours per drive, so a total of 35 hours. I realized I could probably run all 7 processes in parallel since, on my system, the drive controller is a lot faster than any individual drive. So I decided to use the *screen* command and see if I could make that work.

[wipedrives2.sh](#)

```
#!/usr/bin/env bash

for drive in a b c d e f g h
do
    screen -dmS sd$drive bash -c "dd if=/dev/zero | pv -petrs 580G | dd
of=/dev/sd$drive"
```

done

Basically, we're using a bash for loop to grab all the drive names (I just used the last letter), running screen and immediately detaching the new process after telling it to run `bash -c` and the command after it in quotes (so it would not interpret the pipes in our current, non-screen shell). I'm running this right now, and `pv` is predicting it will be done in 11.5 hours, or less than a third of the time. BUT, it is really heating up the office with 7 drives being continuously written to at the same time.

Warning: When SystemRescueCD boots, it tries to assemble any mdadm (software RAID) arrays, and since they are locked, `dd` and `wipefs` won't be able to write to them (maybe). In that case, do the following:

```
# find any mdadm volumes running on Linux
cat /proc/mdstat
# assuming it showed you md127 was running (normal)
mdadm --stop /dev/md127
# it should stop the MD array and make the individual drives accessible
```

Check SSL Cert Expiration Date

Ever wondered when your SMTP SSL Certificates are up for renewal? What DNS entries your certificates have? A quick and dirty way of doing it from the command line was shown at [<https://serverfault.com/questions/131627/how-to-inspect-remote-smtp-servers-tls-certificate#131628>] and [<https://stackoverflow.com/questions/13127352/how-to-check-subject-alternative-names-for-a-ssl-tls-certificate>]

Note: the discussions covered other things, and are well worth a 5 minute read.

This is a quick and dirty that will get the certificate (and a lot of other stuff), but the certificate is in its MIME encoded format.

```
printf 'quit\n' | \
openssl s_client -connect smtp.example.com:25 -starttls smtp
```

This basically makes a connection to `smtp.example.com` on port 25, issuing a `starttls`, then sends the `quit` command which logs out. The `openssl` command retrieves the the entire conversation, which includes the certificate, and displays it on the

You can do the same thing for other ports, like 587 for submission. If you want to test the SSL port (465), just remove the `-starttls smtp` from the command:

```
printf 'quit\n' | \
openssl s_client -connect smtp.example.com:465
```

If you want to test an IMAP server, you need to send it a different logout (the first line). To log out of it, you need `a1 logout` followed by a line return, so

```
printf 'a1 logout\n' | \
```

```
openssl s_client -connect mail.example.com:143 -starttls imap
```

Again, connecting to imaps (port 993), you just don't do the starttls

```
printf 'a1 logout\n' | \  
openssl s_client -connect mail.example.com:143 -starttls imap
```

And, finally, to look at a web site certificate, use port 443, and simply a line return, but you need to put in the server name on systems which have more than one web site (virtual hosting). Do that with the `-servername` flag.

```
printf "\n" | \  
openssl s_client -showcerts -servername web.example.com -connect  
web.example.com:443
```

All the above is well and good, but it would be nice to decode the certificate, wouldn't it? Well, openssl has a command that will allow you to inspect a certificate using the `openssl x509` subcommand. For additional information, see *man openssl-x509*. We want the `-noout` flag to keep our dump clean (prevents the output of the encoded version of the certificate)

Dump the certificate

Turning the certificate into something a human can read is done with the command `-text` flag, so let's pipe the output of the previous command to that.

```
printf 'quit\n' | \  
openssl s_client -connect smtp.example.com:25 -starttls smtp | \  
openssl x509 -text -noout
```

If you want to find what names the certificate is valid for, they are on a line which contains the text DNS, so grepping the output of the above will give you what you need without reading the whole thing.

```
printf 'quit\n' | \  
openssl s_client -connect smtp.example.com:25 -starttls smtp | \  
openssl x509 -text -noout | \  
grep DNS
```

Get Dates

You could use `grep` to find the expiration date of a certificate

```
printf 'quit\n' | \  
openssl s_client -connect smtp.example.com:25 -starttls smtp | \  
openssl x509 -text -noout | \  
grep 'Not After :'
```

But, the openssl x509 has a special flag for that, `-dates`, so it is simpler to write it as

```
printf 'quit\n' | \  
openssl s_client -connect smtp.example.com:25 -starttls smtp | \  
openssl x509 -dates -noout
```

Other

Again, *man openssl-x509* gives you more than I'm showing here under the Display Options section, but just a brief list of some interesting flags. -serial - the serial number of the certificate -subject - Subject Name -issuer - Issuer Name -startdate - beginning date of certificate (notBefore) -enddate - expiry date of certificate (notAfter)

Rename Server

I will occasionally mess up and name a server wrong, then I want to rename it. This is not as simple as it may seem. Most systems have multiple locations you must change, and you might also want to change the ssh host keys, lvm/mdadm/zfs names.

Debian

Following change hostname, mailname (if it exists), etc... Be sure to change *oldname* and *newname* on the sed command.

Note: this assumes the name is unique in the files. So something like 'a' should not be used; manually edit the file.

```
# change the host name, and the postfix name if that is installed  
sed -i.old 's/oldname/newname/g' /etc/hostname /etc/hosts /etc/mailname  
/etc/postfix/main.cf  
/etc/init.d/hostname.sh start  
# update the aliases, if they exist  
newaliases  
# regenerate the ssh keys  
rm -v /etc/ssh/ssh_host_*  
/usr/sbin/dpkg-reconfigure openssh-server
```

Reset Lost Password

The simplest solution is to boot from some kind of live system, then mount the drive and manually edit etc/shadow, which contains a hash of the passwords. In most cases, simply removing the hash sets the user in question to have no password.

We used the SystemRescueCD image (<https://www.system-rescue.org/Download/>) as a bootable USB thumbdrive for this and other purposes.

1. Boot the system from the CD or USB Drive
2. Determine which drive contains the etc/ directory

```
lsblk # Linux
```

3. Mount the drive someplace convenient

```
mount /dev/sda1 /mnt/backup
```

4. Open the shadow file and edit

```
joe /mnt/backup/etc/shadow
```

1. Find the line which contains the user. This is a colon delimited file, with the first column being the username. A sample would look like

```
dailydata:$6$FI3K:18368:0:99999:7:::
```

2. On the line in question, remove everything between the first and second colon. In the sample (which was edited for brevity), it would be `$6$FI3K`. Be sure and DO NOT delete anything else, especially the colons
3. Save the file

5. Reboot the system. The user in question should now be able to log in with no password.

Note: The username in the example is `dailydata`. The password hash is actually very long, in some cases around 100 characters.

If this does not work, you can use the same procedure above but, instead of editing the file directly, mount (as in the above example), then `chroot` into the mounted system and use the `passwd` command. So, after mounting in the above example, do the following:

```
chroot /mnt/backup
passwd root # Change root's password
exit # leave the chroot jail
reboot # or shutdown
```

Wipe Disk Signatures

There are several ways that signatures are included on block devices. You may have a signature saying a device contains an ext4 file system, for example, or is an LVM or RAID member. You can, of course, wipe them all by writing some value to all parts:

```
dd if=/dev/zero of=/dev/sda bs=16M
```

Which will read place the zeros on all blocks of device `sda`. NOTE: `bs=16M` determines how much data is written at one time, and larger numbers greatly increase the speed of the overwrite, up to a significant portion of available RAM.

One of the fastest ways to clean all signatures is the `wipefs` command. It is not as complete as using `dd`, but it is very, very fast, and usually works just fine.

```
# see what the signatures are
wipefs /dev/sda
# remove all of the signatures
```

```
sudo wipefs --all --force /dev/vdb
```

Grab Data via SSH

I needed to grab the output from dmidecode for a bunch of machines. This would have been a good place for something like puppet, but we don't have it fully deployed. However, I have ssh access to most of the machines, so I was trying to figure out how to do it. I wanted the resulting filename to be `hostname -f`.dmidecode, ie the full hostname of the server with .dmidecode at the end. In the following, HOSTNAME is something that ssh can get to.

```
ssh HOSTNAME 'hostname -f ; sudo -S dmidecode' >aeee && FNAM=$(head -1 aeee)
; sed '1d' aeee > $FNAM.dmidecode
```

Dave came up with most of this, then I modified it for my use. Basically, he is returning the hostname and the output of dmidecode to a local temp file named aeee. If that works, then grab the first line into variable FNAM. Then send everything but the first line to the filename \$FNAM.dmidecode.

Note, since dmidecode requires root privileges, I had to use sudo to get it to work. sudo wants a terminal unless you pass the -S parameter, in which case it prints the prompt on STDERR and waits for input. That input is not blanked, so it is visible on your monitor.

Disk Management

Create Swap file

I generally prefer a swap *file* as opposed to a swap *partition*. While swap partitions can be more efficient, swap files are easier to manage (grow/shrink).

This came from <https://www.cyberciti.biz/faq/create-a-freebsd-swap-file/>

```
fallocate -l 4G /swapfile
# if no fallocate on your system, use the following
# dd if=/dev/zero of=/swapfile bs=1024 count=1048576
chmod 600 /swapfile
# use "force" to use the entire "device"
mkswap -f /swapfile
# save, then modify fstab
cp -a /etc/fstab /etc/fstab.save
echo '/swapfile swap swap defaults 0 0' >> /etc/fstab
# turn on swap for everything in fstab
swapon -a
# display the result
swapon --show
```

For BSD (FreeBSD specifically), you create the swapfile with dd, and you must use an md to mount it

```
# create an 8G swapfile
```

```
dd if=/dev/zero of=/swapfile bs=1G count=8
# set permissions very restrictive
chmod 600 /swapfile
# make a copy of fstab, in case we mess something up
cp -a /etc/fstab /etc/fstab.bak
# use mdconfig -lv to find an used md device. In this case, I'm using 42
echo 'md42 none swap sw,file=/swapfile 0 0' >> /etc/fstab
# turn on all defined swap devices
swapon -a
# now list them
swapinfo -g
```

If, as in the case I ran into one time, you have an active swap device you want to get rid of, use `swapinfo` to find it, then use **`swapoff /path/to/device/to/remove`** and remove it from `fstab`

Mount davfs file system

Many web services allow you to mount their contents via davfs. On Linux, it is fairly simple to do this using `davfs2`. **Note** expect this to be slower than what you experience on a LAN. The protocol is generalized, and remember you are doing your work over a connection measured in megabits/second instead of gigabits.

On a [Devuan](#) system, or any Debian derivative, the following will get `davfs2` installed and running. Replace 'jane' below with your username:

```
# Answer yes when asked if unprivileged users should be able to mount
sudo apt -y install davfs2
sudo usermod -aG davfs2 jane
mkdir -p ~/cloud/Tech
mkdir ~/.davfs2
sudo cp /etc/davfs2/secrets ~/.davfs2/secrets
sudo chown jane:jane ~/.davfs2/secrets
chmod 600 ~/.davfs2/secrets
```

Now, you need to edit the secrets file you just copied. You can use an editor, or just append using `echo`. I'm using the latter below. Also, you need to add an entry in `/etc/fstab`.

For the secrets file, you are simply putting in the mount point on your system, a space, the username you will log into the remote machine with, a space, and the password on the remote server.

The `fstab` entry is a standard entry, but uses `davfs` as the type. I chose to have it auto mounted. This is an example for a NextCloud server. Most `davfs` servers have the correct parameters for mounting documented somewhere.

```
# add credentials to .davfs/secrets
# mountpoint username password
echo '/home/jane/cloud/Tech jane your_password_here' >> ~/.davfs2/secrets
# add system mount to /etc/fstab
sudo cp /etc/fstab /etc/fstab.back
sudo echo 'https://cloud.example.com/remote.php/dav/files/jane/Tech
```

```
/home/jane/cloud/Tech davfs user,rw,auto 0 0' >> /etc/fstab
```

Adding a user to a group does not take place immediately; it requires a fresh login. However, you can simulate the login with the following command:

```
sudo su - jane
# Now, you can mount the drive
mount ~/cloud/Tech
# unmounting just uses the standard utilities
umount ~/cloud/Tech
```

Shell (mainly BASH)

Find files within date range containing text

A client needed to find a lost e-mail. All he knew was that it arrived sometime on the 24th of Apr 2020, and who it was from. Not sure if the `-newerct` parameter is available on all versions of `find`, or if it is specific to GNU/Linux.

```
find Maildir -type f -newerct '26 Apr 2022 00:00:00' ! -newerct '27 Apr 2022 00:00:00' -exec grep -il 'from:.*user@example.org' \{\} \;
```

This is very fast, since the `find` command rapidly decreases the number of messages which must be scanned (he has almost 300k e-mails in various folders, and it took less than 2 seconds).

Find newest files in a directory tree

This will go through an entire directory tree under the current directory and locate the newest 5 files.

```
find . -type f -exec stat --format '%Y :%y %n' "{}" \; | sort -nr | cut -d: -f2- | head
```

- Change `find .` to `find /some/path` to change the starting directory
- Change `head` to `head -n 10` to grab the newest 10 files.
- You can add any kind of filter also, so entering `-iname '*.jpg'` after the `-type f` would only find files ending in jpg.

Count all files in directory tree(s)

I was actually using this to count files in a maildir type directory. I needed to know how many total e-mails each user had, then I wanted to know how many they stored in their Inbox.

At a different domain, I needed to know only specific users. They all had account names of the form 'mca-something' so, since 'mca' is pretty uncommon, I just `grep'd` that (could have used `egrep '^mca'` even better, I guess).

Note: this is really not accurate as most IMAP servers store several configuration and control files in the directory, but since that is 2-5 per directory, and I had users storing tens of thousands of e-mails in the Inbox, I didn't break it down any further. You can always look at the Maildir and see some kind of pattern to send to egrep if you want more accuracy.

```
# count all files in all subdirectories
for dir in `ls`; do echo -n " $dir " ; find $dir -type f | wc -l ; done
# count all files in all specific subdirectories identified by a pattern
(mca)
for dir in `ls | grep mca`; do echo -n " $dir " ; find $dir -type f | wc -l
; done
# find inn a subdirectory, ie the Inbox
for dir in `ls`; do echo -n " $dir " ; find $dir/Maildir/cur -type f | wc -l
; done
```

create multiple zero filled files

Sometimes, especially before doing a full disk backup using compression, it is good to write 0's to all unused disk space. This can be done quite easily with a simple dd command (assuming the current directory is on the partition you wish to do this to).

```
dd if=/dev/zero of=./deleteme
rm deleteme
```

This will create a single file, deleteme, which contains nothing but 0's in it (and thus is very compressible), then deletes the file.

However, I have found that I like to have several files which I can then leave on the disk in case I need to perform the copy in the future. I can leave myself plenty of disk space to do my work, and if I need more space, I simply delete some of the files I created. In this case, I'm assuming I have 49.5 gigabytes of free disk space, and I want to zero it all out, then free up 10G for running the system.

```
for i in {01..50} ; do echo Loop $i ; dd if=/dev/zero of=./deleteme.$i bs=1M
count=1024 ; done
for i in {41..50} ; do rm deleteme.$i ; done
```

This will create 50 1 gigabyte files in the current directory, each filled with zeros. Since I am trying to write 50 gigabytes but only have 49.5, the last write will fail since I have no more disk space to write to.

I then delete the last 10 files I created, which gives my system some space to run in.

break a file apart into pieces

In many cases, you have to take a large file and break it into smaller pieces. In this case, use the Unix command *split* to do so. In the following example, I'm taking the 23 Gigabyte file and breaking it into 23 1 Gigabyte files, with numeric suffixes beginning with 07, then 08, all the way to 30.

```
split --suffix-length=2 --bytes=1G --numeric-suffixes=07 --verbose deleteme
deleteme.
```

note that the original file (deleteme) is not modified, so you will need as much space as it occupies, plus a little for overhead.

ssh

Create new key, no passphrase

Create a new rsa key with no passphrase. Useful when you want two machines to talk to each other using automated processes, though it is very insecure if the primary storage is ever disabled.

- -C parameter allows you to define a comment (default is user@hostname)
- -t defines the type of key to create (rsa, dsa, etc...)
- -b is the number of bits to use in the key. Some keys (dsa) have a fixed size. Larger number of bits is harder to crack, but uses more resources.
- -f define the file to create for the private key. The public key will be the same, with .pub added

```
ssh-keygen -t rsa -b 4096 -f id_rsa -C 'new comment for key' -N ''
```

Create missing host keys

Create new host keys (run by root only). When a Unix system is initially set up, several ssh keys are created to identify the system. The following command allows you to change those. The one command will generate all key pairs which are not already created, with an empty passphrase for the private keys.

```
su
# enter root password to become root
ssh-keygen -A
exit # return to unprivileged user
```

Upgrade existing private key storage

Upgrade existing rsa private key to newer storage format. This only affects the encryption on the private key. It does not alter the key at all, so it still works as you are used to

```
ssh-keygen -p -o -f ~/.ssh/id_rsa
```

Change passphrase and/or comment on existing private key

- -p tells it to change the passphrase
- -c tells it to change the comment
- -f tells which file contains the information

```
ssh-keygen -p -c -f ~/.ssh/id_rsa
```

Using multiple key pairs

You can have multiple key pairs for a single user, by simply generating them with different file names, then passing the `-i` (identity) flag on the command line. WARNING: if you mess up the `-f` parameter, you can end up overwriting your default, which is stored as `id_rsa` (or something similar), so back up your stuff first. The following example assumes `rsa`.

```
# make a copy in case we mess up
cp ~/.ssh/id_rsa ~/.ssh/id_rsa.original
cp ~/.ssh/id_rsa.pub ~/.ssh/id_rsa.pub.original
# generate two new keys for two separate applications
ssh-keygen -t rsa -b 4096 -f id_rsa.server1 -C 'key for server1' -N
'passphrase for this key'
ssh-keygen -t rsa -b 4096 -f id_rsa.server2 -C 'key for server2' -N
'passphrase for this key'
```

Copy `id_rsa.server1.pub` to `server1:~/.ssh/authorized_keys`, and `id_rsa.server2.pub` to `server2:~/.ssh/authorized_keys`

To go to a machine named `server`, which uses the default, simply execute

```
ssh server
```

however, to go to `server1`, using its separate key pair

```
ssh -i "~/.ssh/id_rsa.server1" server1
```

and do something similar for `server2`

See config file section for a way to make it easier

using the config file

There are two files which, by default, allow you to make life easier on yourself when using `ssh`. `~/.ssh/config` is local, and `/etc/ssh/ssh_config` is global. The global file location may be different for other operating systems, but I haven't run into that yet.

We'll concentrate on the local config file. Basically, this is a standard text file, with restrictive permissions (0600). The file contains a stanza which begins with the keyword `Host` (case insensitive), followed by multiple line which set parameters for `ssh` when called. Each line is a keyword, as space, and a value.

[config.example](#)

```
Host myshortname
HostName realname.example.com
```

```
# use this for myother server
Host myother realname2.example.org
  HostName realname2.example.org
  IdentityFile ~/.ssh/realname2_rsa
  User remoteusername
  Port 43214
```

The example lists two entries. Note that whitespace is ignored, so indentation is done in the second one to make it easier to read by humans.

The first stanza simply creates an alias (myshortname) for a connection. Issuing the command

```
ssh myshortname
```

uses the default identity file (`~/.ssh/id_rsa`), username (your current user name) and port (22) to connect to `realname.example.com`

The second does an override of several parameters. The following two commands are equivalent:

```
ssh myother
# is the same as
ssh -p 43214 -i "~/.ssh/realname2_rsa" remoteusername@realname2.example.org
```

There are pages and pages of options by running the `man ssh_config` command, where you can include other files, set X11 forwarding, basically everything.

NOTE: I especially like this since I always get `ssh -p` and `scp -P` mixed up, and programs which use `ssh` (rsync, etc...) will use this file.

References

- <https://stackoverflow.com/questions/2419566/best-way-to-use-multiple-ssh-private-keys-on-one-client#2419609>
- <https://linuxize.com/post/how-to-add-swap-space-on-debian-9/>
- https://docs.nextcloud.com/server/18/user_manual/files/access_webdav.html
- <https://www.cyberciti.biz/faq/create-a-freebsd-swap-file/>
- <https://www.mybluelinux.com/test-imap-with-telnet/>
- <https://serverfault.com/questions/131627/how-to-inspect-remote-smtp-servers-tls-certificate#131628>

From:

<https://kb.unixservertech.com/> - **Unix Server Tech Knowledge Base**

Permanent link:

<https://kb.unixservertech.com/quickreference/unix?rev=1696969105>

Last update: **2023/10/10 20:18**



