Unix Quick Reference

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

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).

```
fallocate -l 1G /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
mv /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
```

Shell (mainly BASH)

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 a 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 equivilent:

```
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 sh_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/