Reviewing Docker Logs

Many of you know that it’s possible to access Docker container logs using “docker logs” command. But fewer people know that it’s possible to follow logs stream for new messages (like tail -f), and even fewer yet are aware that it’s possible to specify timestamps of the period you want to review – showing only specific logs during that period.

Show Docker Container Logs

Easiest command is this:

 # docker logs mycontainer1

This will show you all the available logs for the mycontainer1, showing most recent logs last.

Following Docker Container Logs

To follow Docker container for new logs that happen after you start the command, use -f option – like this:

docker logs -f mycontainer1

Initially output won't be very different from the first example, without -f. But you won't get command line prompt and if you wait long enough, new log messages will start appearing.

DockerLogs Around Certain Time

This is a true gem – using these options allows you not only to focus on specific time period, but also to automatically extract only certain logs based on their timestamps.

NOTE: It's possible to specify proper timestamps – meaning both date and time, but I find that just the date part is more than enough.

For timestamp based logs access, use the –since and –until options of the **docker logs** command.

Here's an example of looking at a Java based container, as you can see only the messages from March 30th and until April 1st are shown:

root@s2:~ # **docker logs -f confluence --since 2020-03-30 --**

until 2020-04-01
30-Mar-2020 07:52:27.292 INFO [http-nio-8090-exec-7] org.apache.coyote.http11.Http11Processor.service Error parsing HTTP request header
Note: further occurrences of HTTP request parsing errors will be logged at DEBUG level.
java.lang.IllegalArgumentException: Invalid character found in method name. HTTP method names must be tokens
at org.apache.coyote.http11.Http11InputBuffer.parseRequestLine(Http11InputBuffer.java:415)
at org.apache.coyote.http11.Http11Processor.service(Http11Processor.java:292)
at org.apache.coyote.AbstractProcessorLight.process(AbstractProcessorLight.java:66)
at org.apache.coyote.AbstractProtocol$ConnectionHandler.process(AbstractProtocol.java:861)
at org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(NioEndpoint.java:1579)
at org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProcessorBase.java:49)
at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1149)
at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:624)
at org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run(TaskThread.java:61)
at java.lang.Thread.run(Thread.java:748)
30-Mar-2020 09:22:28.099 INFO [http-nio-8090-exec-1] org.apache.coyote.http11.Http11Processor.service Error parsing HTTP request header
Note: further occurrences of HTTP request parsing errors will be logged at DEBUG level.

```
      java.lang.IllegalArgumentException: Invalid character
found in method name. HTTP method names must be tokens
                                                                at
org.apache.coyote.http11.Http11InputBuffer.parseRequestLine(Ht
tp11InputBuffer.java:415)
                                                                at
org.apache.coyote.http11.Http11Processor.service(Http11Process
or.java:292)
                                                                at
org.apache.coyote.AbstractProcessorLight.process(AbstractProce
ssorLight.java:66)
                                                                at
org.apache.coyote.AbstractProtocol$ConnectionHandler.process(A
bstractProtocol.java:861)
                                                                at
org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(N
ioEndpoint.java:1579)
                                                                at
org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProce
ssorBase.java:49)
                                                                at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolEx
ecutor.java:1149)
                                                                at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolE
xecutor.java:624)
                                                                at
org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run
(TaskThread.java:61)
         at java.lang.Thread.run(Thread.java:748)
30-Mar-2020  23:57:24.900  INFO  [http-nio-8090-exec-3]
org.apache.coyote.http11.Http11Processor.service Error parsing
HTTP request header
  Note: further occurrences of HTTP request parsing errors
will be logged at DEBUG level.
      java.lang.IllegalArgumentException: Invalid character
found in method name. HTTP method names must be tokens
                                                                at
org.apache.coyote.http11.Http11InputBuffer.parseRequestLine(Ht
tp11InputBuffer.java:415)
                                                                at
```

```
org.apache.coyote.http11.Http11Processor.service(Http11Process
or.java:292)
                                                    at
org.apache.coyote.AbstractProcessorLight.process(AbstractProce
ssorLight.java:66)
                                                    at
org.apache.coyote.AbstractProtocol$ConnectionHandler.process(A
bstractProtocol.java:861)
                                                    at
org.apache.tomcat.util.net.NioEndpoint$SocketProcessor.doRun(N
ioEndpoint.java:1579)
                                                    at
org.apache.tomcat.util.net.SocketProcessorBase.run(SocketProce
ssorBase.java:49)
                                                    at
java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolEx
ecutor.java:1149)
                                                    at
java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolE
xecutor.java:624)
                                                    at
org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run
(TaskThread.java:61)
        at java.lang.Thread.run(Thread.java:748)
```

That’s it for today, have fun with Docker!

See Also

- Docker software
- Stop all containers
- Restart Docker containers
- Tidy up Docker volumes
- Docker module in Ansible
- Docker Reference

bash history

BASH

Bash is still my **Unix shell** of choice in 2020, and history is its major functionality that I rely on. This post will be an experiment: I want to collect as much useful information as I can remember in a single post about a particular topic. It's a concept very similar to Unix Reference pages, but for smaller topics.

Shell History

Shell history is a functionality that allows you to use history command for quickly accessing the commands you typed in the past:

```
603  exit
604  docker ps
605  systemctl start docker
606  exit
607  docker ps
608  more /etc/redhat-release
```

```
609  systemctl start docker
610  virsh list
611  history
```

Where is the Bash History File?

Bash history is stored in the user's home directory, in the **/home/$USER/.bash_history** file – so if your username is **greys** (like mine!), the location for Bash history is **/home/greys/.bash_history**.

Search Bash History

There's a special command called **history** – you need to learn it. This isn't a separate binary but usually a built-in function in shells like Bash.

You very learn typing "**history**" – this returns all of the commands you've typed before, then start using **history** command with **grep** to find most relevant results, like this:

```
$ history | grep systemctl
582  systemctl restart ssh
583  systemctl restart sshd
586  systemctl restart sshd
605  systemctl start docker
609  systemctl start docker
612  history | grep systemctl
```

… and then progress to REALLY make the most of bash history by using its built-in history management features, like reverse search:

Press **Ctrl+r** right there in the prompt and bash will want you to start typing pattern for searching in its history. It shows you the best matched line from history and if you press Enter you'll execute that command from history again:

```
root@s1:~ #
(reverse-i-search)`systemctl start n': systemctl start nftables
```

In this example above, I pressed Ctlr+r and typed systemctl start n – and bash suggested the "**systemctl start nftables**".

Using Better Timestamps in Bash History

Next evolutionary step is to equip your shell history with proper timestamps – instead of just knowing the order in which your past commands ran, you'll know exactly when this happened:

```
$ export HISTTIMEFORMAT='%F %T '
```

This will give you history like this going forward:

```
$ history |grep systemctl | tail -10
582  2020-03-31 07:07:53 systemctl restart ssh
583  2020-03-31 07:07:53 systemctl restart sshd
586  2020-03-31 07:07:53 systemctl restart sshd
605  2020-03-31 07:07:53 systemctl start docker
609  2020-03-31 07:37:24 systemctl start docker
612  2020-03-31 08:01:15 history | grep systemctl
613  2020-03-31 08:04:26 systemctl start nftables
619  2020-03-31 08:08:08 history |grep systemctl
621  2020-03-31 08:08:16 history |grep systemctl | tail -10
```

IMPORTANT: this will only add meaningful timestamps to current and future commands. All the ones in the past will have no valid timestamp, so they'll just show you the date when you enabled history using **HISTTIMEFORMAT**.

Configure Permanent History Settings

You need to save history settings in your bash profile – this makes sure the same settings are applied next time you SSH login to the server or start another terminal window with Bash prompt on your desktop.

Step 1: check for existing history settings

Check for anything like HIST (not HISTORY because some parameters are shortened) in the **/home/$USER/.bash_profile** file of yours:

```
greys@xps:~ $ grep HIST .bash_profile
export HISTTIMEFORMAT='%F %T '
```

See? If I want to update the **HISTTIMEFORMAT**, I don’t need to add new option to .bash_profile – I’ll just find and edit the existing one.

Step 2: update or add history settings to bash profile

Let’s say I want to update **HISTFILESIZE** parameter to specify maximum size of the .bash_history file. Here’s how I do it (assuming it’s not found in Step 1):

```
greys@xps:~ $ echo "HISTFILESIZE 10000000" >> /home/greys/.bash_profile
```

Temporarily Disable Bash History

If you need to run a few sensitive commands like API keys or passwords that are passed as command line options, it’s probably best NOT to commit them into bash history.

Just unset the **HISTFILE** right from command line before you type anything sensitive:

```
greys@xps:~ $ unset HISTFILE
```

After that your history won’t be committed even when you finish your current Bash session.

IMPORTANT: best do this in brand new Bash session – otherwise you may lost relevant history that happened prior to you unsetting **HISTFILE**.

How To Centralize Bash History

Once timestamps are showing and history is tracking, you will discover a peculiar thing: some servers you’ll be accessing via multiple remote sessions, and their history won’t all necessarily be captured.

To help with this, we need to centralize bash history – that is, configure your bash profile to commit history more often (via PROMPT_COMMAND variable):

```
export HISTCONTROL=ignoredups:erasedups
shopt -s histappend
export
PROMPT_COMAMND="${PROMPT_COMMAND:+$PROMPT_COMMAND$'\n'}history
-a; history -c; history -r;"
export HISTCONTROL=ignoreboth
```

What this will do is:

- enable history file append functionality (shopt -a histappend)
- make sure every bash prompt (new command line) updates

history file and refreshes current history from it (in case other sessions updated history file)

Can you think of any other cool tricks with Bash History? Let me know and I'll update this page.

See Also

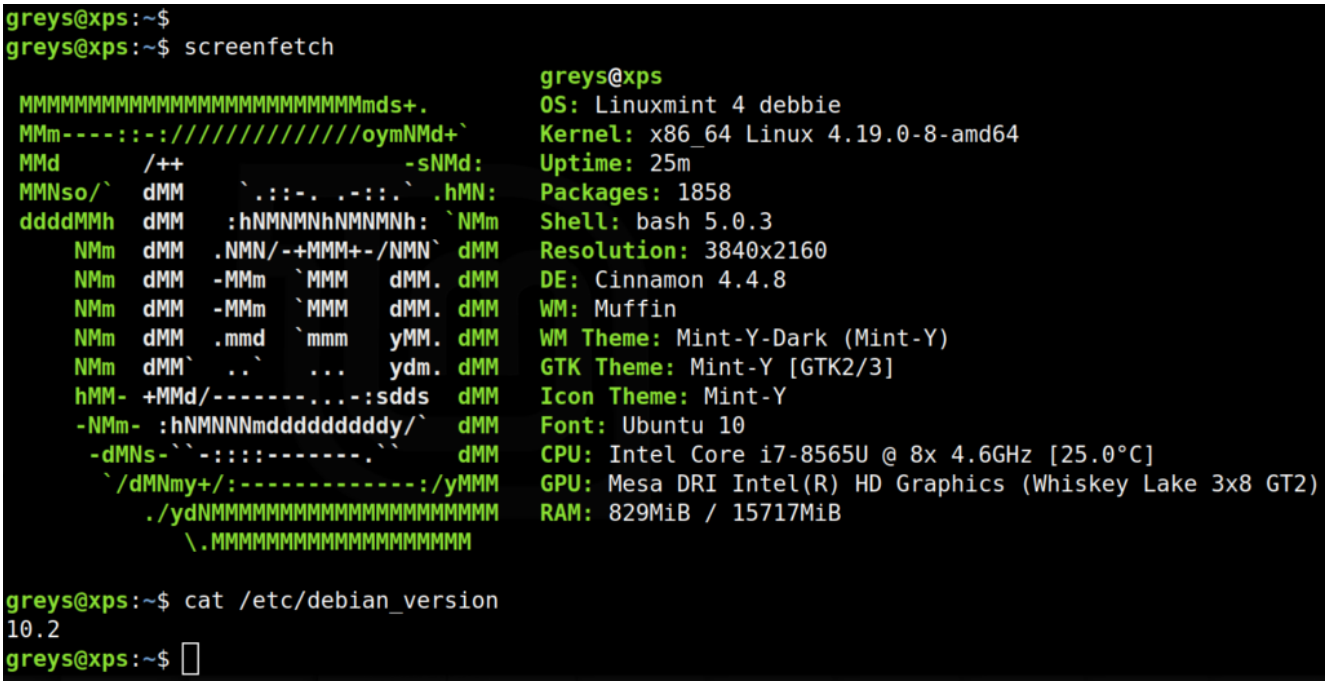
- bash 5.0
- Examples: Bash scripts

---

screenFetch in LMDE4

Got that **Linux Mint Debian Edition 4** installed on my Dell XPS laptop, and it looks and feels amazing!

Here's the screenFetch from it:



More to follow, take care everyone.

See Also

- Linux Mint
- Linux Mint Debian Edition 4
- Linux Mint 19.2 Released

All AWS Certification Exams Can Be Taken Online Now

AWS

Just as I was about to pick a date for my **AWS Certified SysOps Administrator (Associate)** exam, a bit of good news arrived: it's possible to take this and any other AWS certification exam online now.

How To Take AWS Certification Exam Online

In the update from March 30, 2020 AWS Certification FAQ confirms that **you can now take all AWS Certification exams with online proctoring**:

I just checked and there doesn't seem to be any queue for at least the **SysOps Administrator** exam I need. There are available slots for tomorrow and pretty much any day of the next month.

Proctored exams allow you to obtain a certification from your home or home office: you just need a few things to quality.

- stable Internet connection
- webcam (you and your room will be watched as you're taking the exam)
- a room where you can be alone for the duration of the exam

Pearson Vue have a special test available to confirm your computer system is suitable for taking the exam, you can give it a try here.

See Also

- Get AWS instance info with ec2-metadata

AttributeError: module has no attribute in Python

Python

One very common mistake almost everyone makes in **Python** is this: you import a module for some additional functionality, but **Python** won't interpret your code and instead will return you an **AttributeError** message.

AttributeError: module 'csv' has no attribute 'reader'

I needed to parse a CSV file, so I created a new file for the Python code using vim editor:

```
greys@mcfly:~ $cd ~/proj/python
greys@mcfly:~/proj/python $ vim csv.py
```

with the following contents:

```
!/usr/local/bin/python3

import csv

with open('input-data/sample.csv', newline='') as csvfile:
     sample_report = csv.reader(csvfile, delimiter=' ',
quotechar='|')

    for row in sample_report:
        print(', '.join(row))
```

When I attempted to run this code, I received an error:

```
greys@mcfly:~/proj/python $ chmod a+rx csv.py
```

```
greys@mcfly:~/proj/python $ ./csv.py
Traceback (most recent call last):
File "./csv.py", line 3, in
import csv
File "/Users/greys/Documents/proj/python/csv.py", line 6, in
face_report  =  csv.reader(csvfile,  delimiter='  ',
quotechar='|')
AttributeError: module 'csv' has no attribute 'reader'
```

csv is a standard module in the Python 3 library, so this puzzled me a bit. It definitely has the **reader** method, as examples on many websites show.

So Why This AttributeError Message?

Having double-checked for typos and indentation, I realised that this mistake is probably related to my filename.

You see, what's happening is that my code tries to import Python module named csv. It's a standard module, but purist approach insists on not making such assumptions. That's why the Python interpreter checks current directory for any modules named csv before it goes searching further in its standard locations.

Because my own example code was saved in the csv.py file, I was making it import itself instead of the standard (global) Python 3 csv module.

Solution for this Type of AttributeError

The fix is simple: rename your file to something else and it will no longer be importing itself. Python will not find any modules with specified name in your current directory and will then assume you're definitely talking about a module from standard library.

Have a look, once I renamed the file it started working:

```
greys@mcfly:~/proj/python $ mv csv.py csv-test.py
greys@mcfly:~/proj/python $ ./csv-test.py
username,userid,fullname,homedir,password_hash
```

Hope this saves you time some day, enjoy!

See Also

- Check Python Version
- Converting Epoch Time with Python
- Book Review: Practical Programming Python
- Setting Alternatives Path for Python in RHEL8
- Book review: Introduction to Computer Science with Python
- Book Review: Text Processing with Python

Copy Disk Partition with sfdisk

I’ve just learned about a very old but pretty cool Linux command – **sfdisk**. Somehow I’ve always managed to use **fdisk** and **parted**, but **sfdisk** is also very useful. Specifically, I’ve learned to use it for copying partition of one (old) disk to new (replacement) disk.

Confirm Disk Partition Layout

```
root@ubuntu:~ # sfdisk -d /dev/sda
 label: dos
 label-id: 0xc45b9a38
 device: /dev/sda
 unit: sectors
 /dev/sda1 : start=        2048, size=   134213632, type=83, bootable
```

New disk (for this example today) didn’t have any partitions:

```
root@ubuntu:~ # sfdisk -d /dev/sdb sfdisk: /dev/sdb: does not contain a recognized partition table
```

Save Disk Partition Layout into File

Very simple command to save layout into **partitions.txt** file:

```
root@ubuntu:~ # sfdisk -d /dev/sda > partitions.txt
```

Apply Partition Layout from File to Disk

And this is how I can apply the layout from file **partitions.txt** to the new disk **/dev/sdb**:

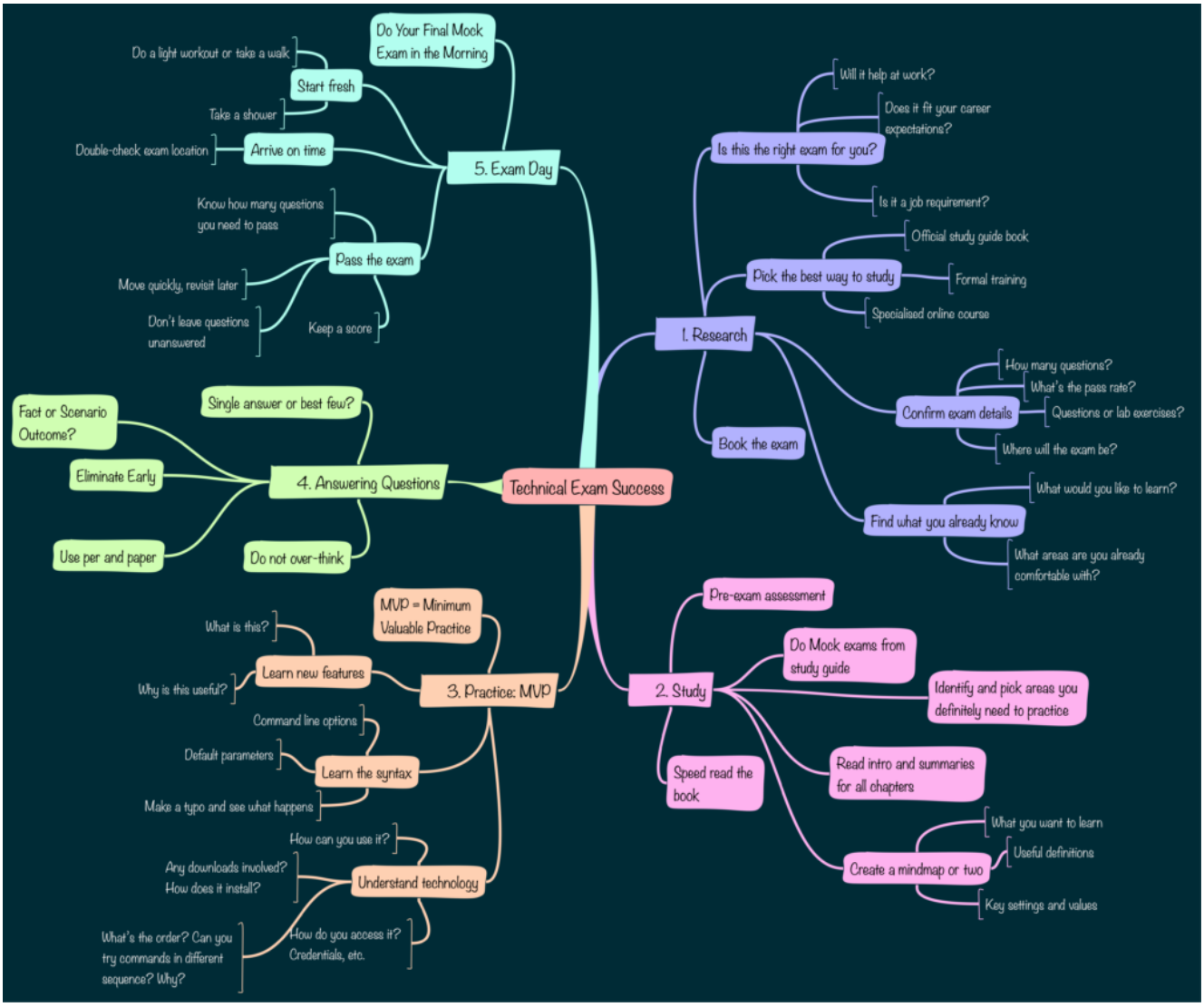
```
root@ubuntu:~ # sfdisk -f /dev/sdb < partitions.txt
 Checking that no-one is using this disk right now … OK
 Disk /dev/sdb: 5 GiB, 5368709120 bytes, 10485760 sectors
 Disk model: Ubuntu Linux-1 S
 Units: sectors of 1 * 512 = 512 bytes
 Sector size (logical/physical): 512 bytes / 4096 bytes
 I/O size (minimum/optimal): 4096 bytes / 4096 bytes
 Disklabel type: dos
 Disk identifier: 0xc45b9a38
 Old situation:
 Device     Boot Start      End  Sectors Size Id Type
 /dev/sdb1  *     2048 10485759 10483712   5G 83 Linux
               Script header accepted.
       Script header accepted.
       Script header accepted.
       Script header accepted.
        Created a new DOS disklabel with disk identifier
0xc45b9a38.
       /dev/sdb1: Created a new partition 1 of type 'Linux'
```

```
and of size 5 GiB.
       /dev/sdb2: Done.
 New situation:
 Disklabel type: dos
 Disk identifier: 0xc45b9a38
 Device     Boot Start      End  Sectors Size Id Type
 /dev/sdb1  *     2048 10485759 10483712   5G 83 Linux
 The partition table has been altered.
 Calling ioctl() to re-read partition table.
 Syncing disks.
```

See Also

- tune2fs command
- change filesystem label with tune2fs

Technical Exam Success – Mindmap



Technical Exam Success – Mindmap

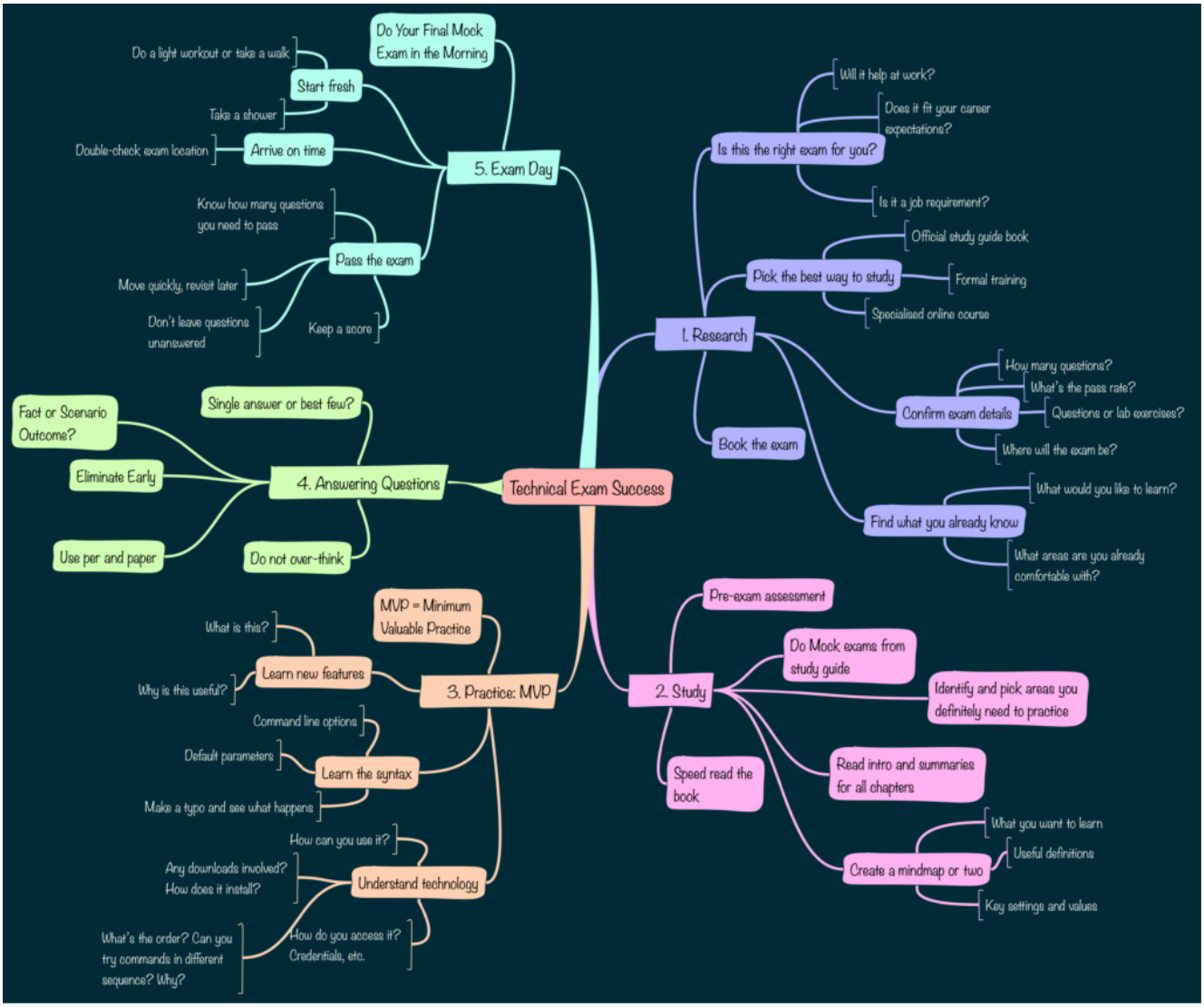
I'm writing an e-book on my approach to taking technical exams and obtaining relevant certifications.

There's going to be two versions of the book:

- **Technical Exam Success – Checklist** – mobile friendly, brief summary of improving your chances (you can get it by subscribing to my mailing list in sidebar)
- **Technical Exam Success** – full edition of the book with examples and thorough description of techniques (this is a link to Leanpub placeholder, more information to follow)

Today I just wanted to share a mindmap based on the checklist – it's a great way to structure your approach for any technical certification.

Click on the image below for the high resolution image:



Have a look and let me know what you think!

See Also

- Unix Book Reviews
- Unix Tutorial Projects
- Basic Unix Commands
- Unix Commands

Connect to VPN from macOS Command Line

macOS

I knew it's possible to setup VPN connections using native System Preferences in macOS, but discovered recently that it's possible to manage them from command line.

Today I'll just show you how to connect or disconnect from an existing VPN profile – perhaps some other time I'll explain how to setup VPN profile and connection parameters from

scratch.

Reviewing Network Connections in System Preferences

You can find existing connections – networked, wireless, bluetooth and VPN – in Network section of the System Preferences app:

You can simply click the Connect button to activate VPN connection if you're already there in the settings screen. But using command line it's possible to activate the same VPN connection without ever opening System Preferences app or selecting the profile using keyboard and mouse.

Activate VPN Connection using networksetup

networksetup command lets you manage lots of network connection aspects.

Here's a command to activate VPN connection for my profile **HomeVPN**:

greys@maverick:~ $ **networksetup -connectpppoeservice "HomeVPN"**

and although it’s possible to review VPN connection status in **System Preferences**:

… it’s also possible to check it from the command line:

greys@maverick:~ $ **networksetup -showpppoestatus "HomeVPN"**
connecting

and then – eventually – confirm it’s connected:

greys@maverick:~ $ **networksetup -showpppoestatus "HomeVPN"**
connected

See Also

- macOS
- macOS Catalina
- install Jekyll 4 macOS
- List kernel modules in macOS
- Show Kernel extension files in macOS

SSH client Config for Using a Jumphost

```
Host s3
        Hostname s3.unixtutorial.org
        Port 212
        ProxyCommand ssh -W %h:%p -q greys@gw.ts.fm -p 202
```

SSH jumphost configuration in .ssh/config

I needed to use a mobile 4G hotspot today and realised there's another very common reason for using SSH jumphosts. You see, when I'm on a 4G hotspot I tend to use VPN client for securing Internet connection. And jumphosts may help in VPN scenarios.

Sometimes I then switch to home WiFi and need to manually disable VPN, otherwise my connection keeps getting protected. Using VPN on home WiFi allows me to use SSH client on laptop to use one of my Raspberry Pi servers as jumphost for connecting to external servers.

The reason this works is because laptop is the only system running VPN, but it usually still has access to local networks in my home network. So if I ssh onto one of my Raspberry Pi servers, any connection I make from there on will use my home's broadband IP (because Raspberry doesn't have VPN configured).

How To Specify Jumphost in SSH Client Config

I have my dedicated servers configured like this in /Users/greys/.ssh/config file:

```
Host s3
     HostName s3.unixtutorial.org
     Port 212
```

If I need to use jumphost to access s3, I'll update this configuration setting:

```
Host s3
      HostName s3.unixtutorial.org
      Port 212
      ProxyCommand ssh -W %h:%p greys@gw.ts.fm -p 202
```

Just to remind you, gw.ts.fm is my SSH jumphost name and it's listening to SSH on port 202.

In case you need to edit this config file in Linux, this is in /home/greys/.ssh/config file for me – so /home/$USER/.ssh/config format.

See Also

- SSH reference

- Ansible via SSH jumphost
- SSH port
- **SSH port forwarding**
- Ansible non-standard port

How To Use Ansible with SSH Jumphost

Red Hat Ansible

There are many scenarios where some of your infrastructure isn't directly accessible from your **Ansible** deployment system (home desktop or work laptop). That's why I think it's super useful to know how you can get certain **Ansible** hosts use SSH jumphost – a special server that accepts your connection and forwards it to those remote systems that are not accessible directly.

Why You May Need an SSH Jumphost

Here are just some of the most common scenarios. If you know of other situations – please get in touch as I'd like to explore them.

- **On-line Infrastructure Behind Firewall** – especially in online services companies, most of servers are not accessible directly. Typically you'll have a gateway server (SSH jumphost) or VPN to access such infrastructure.
- **Cloud Infrastructure like AWS with internal networking** – lots of cloud deployments use internal networking that's rarely exposed. Access to hosted apps and services is done via endpoints like load balancers. If you need direct access, a VPN instance or SSH server will be required.
- **Dynamic IP address on Your Workstation** – if you're working from laptop and using 4G or non-standard WiFi hotspots, your public IPv4 address will be different every time. Even if your infrastructure has SSH entry points, they're probably accepting connections only from a list of static and well-defined IP addresses or ranges. You may have some VPS server with static IP address online, that you connect from your laptop and then connect to the infrastructure itself.

How To Configure Ansible Hosts for SSH Jumphosts

Simply add like like this to the host_vars/HOSTNAME.yaml file

for your hostname or – more likely – add the same line to group_vars/GROUP.yaml (obviously replace HOSTNAME with a specific name and GROUP with your server group name):

```
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q greys@gw.ts.fm -p 202"'
```

In this code snippet, this is what everything means:

- **-o ProxyCommand** is a command line option to SSH client
- -W %h:%p – special mode that requests SSH client to forward input/output from the specified SSH client (%h – hostname) and port (%p). You can't see them here, because we're updating Ansible settings. But you'll see it in the example below.
- **greys** is my username
- **gw.ts.fm** is the SSH jumphost I use
- **202** is the SSH port number on my jumphost

What the above configuration does is it makes Ansible use an SSH client command line similar to this (where **myserver** is the remote host we can't access directly):

```
$ ssh -o ProxyCommand="ssh -W %h:%p -q greys@gw.ts.fm -p 202"' myserver1
```

That's it for today! Enjoy!

See Also

- Ansible Software
- Ansible – use custom port
- Support Ansible in SUDO
- **Ansible Reference**