On this page

Linux cheat sheet ALL TIERS SELF-MANAGED

This is the GitLab Support Team's collection of information regarding Linux, that they sometimes use while troubleshooting. It is listed here for transparency, and it may be useful for users with experience with Linux. If you are currently having an issue with GitLab, you may want to check your support options first, before attempting to use this information.

> ⚠ It is beyond the scope of GitLab Support to assist in systems administration. GitLab administrators are expected to know these commands for their distribution of choice. If you are a GitLab Support Engineer, consider this a cross-reference to translate `yum` -> `apt-get` and the like.

Most of the commands below have not been labeled as to which distribution they work on. Contributions are welcome to help add them.

System Commands

Distribution Information

```
# Debian/Ubuntu
uname -a
lsb_release -a

# CentOS/RedHat
cat /etc/centos-release
cat /etc/redhat-release

# This will provide a lot more information
cat /etc/os-release
```

Shut down or Reboot

```
shutdown -h now
reboot
```

Permissions

```
# change the user:group ownership of a file/dir
chown root:git <file_or_dir>

# make a file executable
chmod u+x <file>
```

Files and directories

```
# create a new directory and all subdirectories
mkdir -p dir/dir2/dir3

# Send a command's output to file.txt, no STDOUT
ls > file.txt

# Send a command's output to file.txt AND see it in STDOUT
ls | tee /tmp/file.txt

# Search and Replace within a file
sed -i 's/original-text/new-text/g' <filename>
```

See all set environment variables

```
env
```

Searching

File names

```
# search for a file in a filesystem
find . -name 'filename.rb' -print

# locate a file
locate <filename>

# see command history
history

# search CLI history
<ctrl>-R
```

File contents

```
# -B/A = show 2 lines before/after search_term
grep -B 2 -A 2 search_term <filename>

# -<number> shows both before and after
grep -2 search_term <filename>

# Search on all files in directory (recursively)
grep -r search_term <directory>

# search through *.gz files is the same except with zgrep
zgrep search_term <filename>

# Fast grep printing lines containing a string pattern
fgrep -R string_pattern <filename or directory>
```

CLI

```
# View command history
history

# Run last command that started with 'his' (3 letters min)
!his

# Search through command history
<ctrl>-R

# Execute last command with sudo
sudo !!
```

Managing resources

Memory, Disk, & CPU usage

```
# disk space info. The '-h' gives the data in human-readable values
df -h

# size of each file/dir and its contents in the current dir
du -hd 1

# or alternative
du -h --max-depth=1

# find files greater than certain size(k, M, G) and list them in order
# get rid of the + for exact, - for less than
find / -type f -size +100M -print0 | xargs -0 du -hs | sort -h

# Find free memory on a system
free -m

# Find what processes are using memory/CPU and organize by it
# Load average is 1/CPU for 1, 5, and 15 minutes
top -o %MEM
top -o %CPU
```

Strace

```
# strace a process
strace -tt -T -f -y -yy -s 1024 -p <pid>

# -tt   print timestamps with microsecond accuracy

# -T    print the time spent in each syscall

# -f    also trace any child processes that forked

# -y    print the path associated with file handles

# -yy    print socket and device file handle details

# -s    max string length to print for an event

# -o    output file

# run strace on all puma processes
ps auwx | grep puma | awk '{ print " -p " $2}' | xargs strace -tt -T -f -y -yy -s 1024 -o /tmp/puma.txt
```

Be aware that strace can have major impacts to system performance when it is running.

Strace Resources

- See the strace zine for a quick walkthrough.
- Brendan Gregg has a more detailed explanation of how to use strace.
- We have a series of GitLab Unfiltered videos on using strace to understand GitLab.

The Strace Parser tool

Our strace-parser tool can be used to provide a high level summary of the `strace` output. It is similar to `strace -C`, but provides much more detailed statistics.

MacOS and Linux binaries are available, or you can build it from source if you have the Rust compiler.

How to use the tool

First run the tool with no arguments other than the strace output filename to get a summary of the top processes sorted by time spent actively performing tasks. You can also sort based on total time, # of system calls made, PID #, and # of child processes using the `-S` or `--sort` flag. The number of results defaults to 25 processes, but can be changed using the `-c/--count` option. See `--help` for full details.

```
$ ./strace-parser strace.txt

Top 25 PIDs
-----------

  pid           active (ms)  wait (ms)  total (ms)   % active    syscalls
  ----------    ----------  ---------   ---------   ---------   ---------
  8795             689.072   45773.832   46462.902     16.89%       23018
  13408            679.432   55910.891   56590.320     16.65%       28593
  6423             554.822   13175.485   13730.308     13.60%       13735
...
```

Based on the summary, you can then view the details of system calls made by one or more processes using the `-p/--pid` for a specific process, or `-s/--stats` flags for a sorted list. `--stats` takes the same sorting and count options as summary.

```
$ ./strace-parse strace.text -p 6423

PID 6423
13735 syscalls, active time: 554.822ms, total time: 13730.308ms

  syscall              count        total         max         avg         min  errors
                                     (ms)        (ms)        (ms)        (ms)
  ---------------   --------    ----------  ----------  ----------  ----------  --------
  epoll_wait             628     13175.485      21.259      20.980       0.020
  clock_gettime         7326       199.500       0.249       0.027       0.013
  stat                  2101       110.768      19.056       0.053       0.017  ENOENT: 2076
  ...
  ---------------

  Parent PID: 495
  Child PIDs:  8383, 8418, 8419, 8420, 8421

  Slowest file access times for PID 6423:

     open (ms)        timestamp              error     file name
  -----------   ---------------    ---------------     ----------
      29.818    10:53:11.528954                        /srv/gitlab-data/builds/2018_08/6174/954448.log
      12.309    10:53:46.708274                        /srv/gitlab-data/builds/2018_08/5342/954186.log
      0.039     10:53:49.222110                        /opt/gitlab/embedded/service/gitlab-
rails/app/views/events/event/_note.html.haml
      0.035     10:53:49.125115                        /opt/gitlab/embedded/service/gitlab-
rails/app/views/events/event/_push.html.haml
  ...
```

In the example above, we can see that file opening times on `/srv/gitlab-data` are extremely slow, about 100X slower than `/opt/gitlab`.

When nothing stands out in the results, a good way to get more context is to run `strace` on your own GitLab instance while performing the action performed by the customer, then compare summaries of both results and dive into the differences.

Stats for the open syscall

Rough numbers for calls to `open` and `openat` (used to access files) on various configurations. Slow storage can cause the dreaded `DeadlineExceeded` error in Gitaly.

Also see this entry in the handbook for quick tests customers can perform to check their file system performance.

Keep in mind that timing information from `strace` is often somewhat inaccurate, so small differences should not be considered significant.

Setup	access times
EFS	10 - 30ms
Local Storage	0.01 - 1ms

Networking

Ports

```
# Find the programs that are listening on ports
netstat -plnt
ss -plnt
lsof -i -P | grep <port>
```

Internet/DNS

```
# Show domain IP address
dig +short example.com
nslookup example.com

# Check DNS using specific nameserver
# 8.8.8.8 = google, 1.1.1.1 = cloudflare, 208.67.222.222 = opendns
dig @8.8.8.8 example.com
nslookup example.com 1.1.1.1

# Find host provider
whois <ip_address> | grep -i "orgname\|netname"

# Curl headers with redirect
curl --head --location "https://example.com"

# Test if a host is reachable on the network. `ping6` works on IPv6 networks.
ping example.com

# Show the route taken to a host. `traceroute6` works on IPv6 networks.
traceroute example.com
mtr example.com

# List details of network interfaces
ip address

# Check local DNS settings
cat /etc/hosts
cat /etc/resolv.conf
systemd-resolve --status

# Capture traffic to/from a host
sudo tcpdump host www.example.com
```

Package Management

```
# Debian/Ubuntu

# List packages
dpkg -l
apt list --installed

# Find an installed package
dpkg -l | grep <package>
apt list --installed | grep <package>

# Install a package
dpkg -i <package_name>.deb
apt-get install <package>
apt install <package>

# CentOS/RedHat

# Install a package
yum install <package>
dnf install <package> # RHEL/CentOS 8+

rpm -ivh <package_name>.rpm

# Find an installed package
rpm -qa | grep <package>
```

Logs

```
# Print last lines in log file where 'n'
# is the number of lines to print
tail -n /path/to/log/file
```

Help & feedback

Docs

Edit this page to fix an error or add an improvement in a merge request.
Create an issue to suggest an improvement to this page.
Show and post comments to review and give feedback about this page.

Product

Create an issue if there's something you don't like about this feature.
Propose functionality by submitting a feature request.
Join First Look to help shape new features.

Feature availability and product trials

View pricing to see all GitLab tiers and features, or to upgrade.
Try GitLab for free with access to all features for 30 days.

Get Help

If you didn't find what you were looking for, search the docs.

If you want help with something specific and could use community support, post on the GitLab forum.

For problems setting up or using this feature (depending on your GitLab subscription).

Request support