Command Ninja Documentation

Release 0.1.b

Abhijeet Kasurde

Mar 13, 2017

Contents

1 Command 3
2 Alias 7
3 Bashrc 9
4 Grep 11
5 Find 13
6 sed 15
7 git 17
8 Dpkg 19
9 Apt-get 21
10 rpm 23
11 ssh 25
12 Indices and tables 27

Command Ninja is collection of frequently used Linux and other operating systems' commands which help you to do work more efficiently and swiftly.

Contents:

CHAPTER 1

Command

At Shell Prompt

- Running previous command with sudo:

  ```
  ls -la
  sudo !!
  ```

  Usage: This will run *ls -la* command using sudo command

- Using !! on shell prompt:

  ```
  ls -la a
  echo !!:2
  ```

  Usage: This will print *a* on screen.

- Using !! to replace some part of command:

  ```
  echo "foo"
  !!:gs/foo/bar
  ```

  Usage: Runs previous command replacing foo by bar every time that foo appears

- whoami in mysql database:

  ```
  mysql> select user(),current_user();
  ```

Perl

- Edit file contents:

```
perl -pi.bak -e "tr/[a-z]/[A-Z]/" sample.txt
```

Usage: Creates backup of original file and replace text in given file.

- Replace text in file and creating backup of file:

```
perl -pi.bak -e '<replacement>' <filename>
```

Usage: *perl -pi.bak -e 'tr/[a-z]/[A-Z]/g' sample.txt*

This will convert text of sample.txt to uppercase and create backup of sample.txt as sample.txt.bak

Process

- Find infomartion about process by name:

```
ps ax | grep <process_name>
ps ax | grep <process_name> | grep -v grep
```

Usage:

  – *ps ax | grep apache2*
  – *ps ax | grep [Aa]pache2* #This will only show apache process (without grep)
  – *ps ax | grep apache2 | grep -v grep*

Networking

- Shows the network connections:

```
ifconfig
```

- Show ip address for all available interfaces:

```
ip addr | grep inet
```

- Start webserver serving current directory tree at http://localhost:8000/:

```
python -m SimpleHTTPServer
```

- Find who owns port number:

```
fuser -v -n tcp <port_number>
```

Usage: *fuser -v -n tcp 6000*

Kernel and Kernal module

- Shows the status of modules in the Linux Kernel:

```
lsmod
```

- Shows the list of modules for Linux Kernel:

```
ls /lib/modules/$(uname -r) -R  | grep ko
```

- Finding hardware information:

```
lshw
```

Usage: *lshw -c video* #This will provide information about *video* hardware.

Note: You need to be super-user to run this command and get all information about hardware.

- Show information about particular module:

```
modinfo <driver_name>
```

Usage : *modinfo nvidia* #This will provide information about nvidia driver

Swap

- More swap with a swap file:

```
dd if=/dev/zero of=/swapfile bs=1024 count=65536 #Create 64MB swap file on your␣
↪root partition
mkswap /swapfile 65536          #convert file to swap file
sync
swapon /swapfile        #add swapfile to your swapspace
```

Memory

- Check memory status of:

```
head -2 /proc/meminfo
```

Usage: provide Available and Used memory of current state of machine

- View the different caches and their sizes:

```
sudo head -2 /proc/slabinfo; sudo cat /proc/slabinfo | egrep dentry\|inode
```

Courtesy: `http://linuxaria.com/howto/linux-memory-management`

wget

- Download list of files using *wget*:

```
cat <file_containing_url> | xargs wget -c
```

Usage : *cat urlist.txt | xargs wget -c* #download content of url for urlist.txt

CHAPTER 2

Alias

Following are some alias to make you work faster

- Alias for *cd* command:

```
alias cd..="cd .."
alias 2..="cd ../.."
alias 3..="cd ../../.."
alias 4..="cd ../../../.."
alias 5..="cd ../../../../.."
```

- Alias for *apt-get* command:

```
alias agi='apt-get install'
alias agu='apt-get update'
alias ags='apt-cache search'
alias agsh='apt-cache show'
alias agr='apt-get remove'
alias agd='apt-get dist-upgrade'
```

- Alias for *ssh* command:

```
alias dev="user@dev.example.com -p 8000"
```

- Alias for ‘df’ command:

```
alias df="df -TPh"
```

- Alias for Port usages:

```
sudo lsof -i -P -sTCP:LISTEN
```

CHAPTER 3

Bashrc

- Function to add alias of command:

```
function addalias(){
   echo -e "adding alias $1 for $2\n"
   echo -e "alias ${1}=\"${2}\"" >> ~/.bashrc
   . ~/.bashrc
}
```

Usage : addalias <nameofalias> "<command>"

- Function to find newly installed python module working:

```
function import(){
   python -c "import ${1}" 2> /dev/null
   [ $? -eq 0 ] && echo -e "Python module ${1} installed properly" || \
   echo -e "Python module ${1} is missing or perhaps mispelled"
}
```

Usage : import <name_of_python_module>

- Function to mkdir and cd into directory:

```
function mkcd () {
  mkdir -p "$@" && eval cd "\"\$$#\"";
}
```

Usage : *mkcd abc*

- Function to show IP of all connections

```
function ips () {
        local interface=""
        local types='vmnet|en|eth|vboxnet|wlan|wl|tap|tun'
        local i
                for i in $(
                        ifconfig \
            | egrep -o '(^('$types')[0-9]|inet (addr:)?([0-9]+\.){3}[0-9]+)' \
            | egrep -o '(^('$types')[0-9]|([0-9]+\.){3}[0-9]+)' \
        | grep -v 127.0.0.1
                ); do
                if ! [ "$( echo $i | perl -pi -e 's/([0-9]+\.){3}[0-9]+//g' )" ==
↪"" ]; then
                        interface="$i":
                else
                        echo $interface $i
                fi
                done
}
```

Usage: *ips*

- Function to check “Are you root ?”:

```
chk_root () {
        if [ ! $( id -u ) -eq 0 ]; then
                echo -e "Must be run as root"
                exit
        fi
}
```

Usage: Just use *chk_root* in script

CHAPTER 4

Grep

- Search file for keyword *Error*:

```
grep "Error" mylogfile.log
```

- Search file for keyword *Error* with case-insensitivity

```
grep -i "error" mylogfile.log
```

- Searching several words in file

```
grep -Ei "error|exception|fatal" mylogfile.log
```

- See more after and more before keyword in file:

```
grep -A 10 -B 20 "exception" mylogfile.log
```

  Above grep will show 10 line after and 20 line before *exception* word in *mylogfile.log*

- Search file and line with filename:

```
grep -nrH MyMethodName *
```

  Above grep command will search files in current directory recursively with line and filename.

- Search keyword in file and print only filename:

```
grep -ril <keyword> <location>
```

  Usage: grep -ril "*myword*" .

- Find keyword in file and print filename only

```
grep -l "word" *
```

CHAPTER 5

Find

- Find files with particular word in its name:

```
find <location> -iname <word>
```

Usage : *find /etc -iname "*apache*"*

- Find files with size more than certain size:

```
find <location> -type f -size +<size_integer>M
```

Usage : *find / -type f -size +100M*

- Find files which are not modified in last x number of days:

```
find <location> -mtime +<integer>
```

Usage : *find . -mtime +2* #Find files in current dir which are not modified last 2 days

- Find files which are modified in last x number of days:

```
find <location> -mtime -<integer>
```

Usage : *find . -mtime -2* #Find files in current dir which are modified last 2 days

- Finding image files in current directory on the basis of mime-type:

```
find . -type f -exec file {} \+ | grep -c -i 'image'
```

is faster than:

```
find . -type f -print0 | xargs -0 file -i | grep -i image | wc -l
```

CHAPTER 6

sed

- Match whole string:

```
sed "s/\b<keyword\b/g"
```

Usage: echo “bar foobar” | sed “s/\bbar\b/nobar/g”

CHAPTER 7

git

Following are some alias to make you work faster for *git*

- Undo a git add - removing files staged for a git commit:

  ```
  git reset HEAD <file_name>
  ```

  or:

  ```
  git rm --cached <file_name>
  ```

  Usage: *git reset HEAD a.sh* or *git rm –cached a.sh* This command will remove a file named a.sh from the current index, the “about to be committed” area, without changing anything else.

CHAPTER 8

Dpkg

Quick cheat sheet you will find handy while using dpkg at shell prompt:

- Installing deb file using dpkg:

  ```
  dpkg -i <package_name>.deb
  ```

  Usage: *dpkg -i apache2_2.2.17-1ubuntu1.5_i386.deb*

- Installing deb packages recursively from given directory:

  ```
  dpkg -R <path_to_directory>
  ```

  Usage: *dpkg -R /var/cache/apt/archives/*

- Find all files related to package:

  ```
  dpkg -L <package_name>
  ```

  Usage: *dpkg -L apache2*

- List all package by name:

  ```
  dpkg -l | grep <package_name>
  ```

  Usage: *dpkg -l | grep apache2*

- Find which package is related to particular file:

  ```
  dpkg -S <file_name>
  ```

  Usage: *dpkg -S /etc/apache2/apache2.conf*

- Find status of package:

  ```
  dpkg -s <package_name> | grep Status
  ```

  Usage: *dpkg -s apache2 | grep Status*

- Display details about package package group, version, maintainer, Architecture, display depends packages, description etc.:

  ```
  dpkg -p <package_name>
  ```

  Usage: *dpkg -p apache2*

- List files provided by given package:

  ```
  dpkg -c <deb_package_name>
  ```

  Usage: *dpkg -c apache2_2.2.17-1ubuntu1.5_i386.deb*

- List individual package name installed with short description:

  ```
  dpkg -l <package_name>
  ```

  Usage: *dpkg -l apache2*

- List all package name installed with short description:

  ```
  dpkg -l
  ```

  Usage: *dpkg -l*

- Remove pacakge:

  ```
  dpkg -r <package_name>
  ```

  Usage: *dpkg -r apache2*

- Remove package with all configuration:

  ```
  dpkg -P <package_name>
  ```

  Usage: *dpkg -P apache2*

CHAPTER 9

Apt-get

Quick Guide to *apt-get*

- apt-get update

  This retrieves the current list of packages from all servers in your *sources.list*. If you don't do this from time to time your local list of available packages may become out of date. Do this occasionally before doing a *dist-upgrade* or searching for a new package. The package lists are large: doing an update may result in several MB of data being retrieved from the Internet.

- apt-cache search program

  This will do a keyword search through the list of available packages, including package names and descriptions. You can put in several keywords, for example *apt-cache search text editor* to find a list of text editors.

- apt-cache show program

  Once you've found a package that looks interesting using *apt-cache search*,you can display more detailed information about it using *apt-cache show program*. This will tell you things like the size of the package (important if you are installing it off the Internet) and an extended description, as well as what other packages it depends on in order to work and the name of the developer who maintains the package.

- apt-get install program

  This will get the latest version of the specified package and install it, along with any other packages that it depends on in order to work. If the requested package is already installed, this will upgrade it to the latest available version.

- apt-get remove program

  If you've previously installed a program and decide you don't want it anymore, you can remove it using this command. Because some software packages can depend on others, removing one program may break other programs. Running *apt-get remove* therefore checks first to see if any other software needs the program to work, and uninstalls them as well. This is just one example of the way the Debian package management tools have been designed to try to keep your computer in a sane state, without broken or half-installed software. It's certainly possible to break a Debian system, but generally you have to try to do it. It's unlikely you could do it by mistake.

- apt-get upgrade

  Over time, most of the software packages on your computer will become out of date as new versions are released to add features or fix bugs. You could manually do *apt-get install foo* on each one, but that's not very convenient, so Apt provides a simple way to upgrade your entire system at once. Just type *apt-get upgrade* to have Apt check every single package on your system for a new version, then download and install it. This command will never install new packages, it will only upgrade packages that are already installed.

- apt-get dist-upgrade

  Sometimes you'll have a software package installed, and a new version will come out that has a lot of new features and therefore it now depends on some other program to run. For example, you may have a movie player installed that supports a lot of different movie formats. When new formats come out, modules for those formats may be added in separate packages, so the latest version of the movie player software now depends on a new package that you don't yet have installed on your system. If you just do *apt-get upgrade*, you'll get the latest movie player, but you won't get all the new format packages. The *apt-get dist-upgrade* command solves that problem for you: not only does it get the latest version of every package already installed just like *apt-get upgrade*,it also installs any new packages they need that may not be on your system yet. If you want to keep your system up to date with all the latest updates and security patches, running *apt-get update; apt-get dist-upgrade* from time to time is the best way to do it.

- apt-get clean

  When you ask Apt to install a software package, it downloads the package and stores it in a cache on your disk before it does the actual installation. If you then remove the package, but later change your mind again and re-install it, Apt doesn't need to fetch it off the Internet again because the package is sitting in the local cache. That's great for saving bandwidth, but after a while it can use up space on your disk so it's a good idea to periodically delete old packages from the cache. Running *apt-get clean* will totally flush the package cache, possibly freeing up some precious disk space. Running this command is quite safe, because the worst that can happen is Apt may need to download a package again if you remove it then re-install it.

- apt-get autoclean

  This is almost the same as *apt-get clean*, except it's just a little bit smarter: instead of cleaning out your entire package cache, it deletes only superseded packages. For example, your package cache may contain packages for the last 7 versions of a text editor that has been upgraded a number of times: running *apt-cache autoclean* will delete the oldest 6 versions from the cache, leaving only the latest one. That makes sense because you're not likely to re-install anything except the latest version anyway. This is also a very safe command to run, so if you're a bit tight on disk space and don't want your package cache growing too much you could put it in a Cron job to do an automatic cleanup from time to time. There's really no reason to keep the older packages lying around on disk anyway.

CHAPTER 10

rpm

Quick cheat sheet you will find handy while using rpm at shell prompt:

- Installing rpm package:

```
rpm -ivh <rpm_file>
```

  Usage: *rpm -ivh httpd-2.0.49-4.i386.rpm*

- Upgrage rpm package:

```
rpm -Uvh <rpm_file>
```

  Usage: *rpm -Uvh httpd-2.0.49-4.i386.rpm*

- Remove rpm package:

```
rpm -ev <rpm_file>
```

  Usage: *rpm -ev httpd-2.0.49-4.i386.rpm*

- Remove rpm package without removing dependencies:

```
rpm -ev --nodeps <rpm_file>
```

  Usage: *rpm -ev –nodeps httpd-2.0.49-4.i386.rpm*

- Query all rpm package

```
rpm -qa
```

  Usage: *rpm -qa*

- Query rpm package for short description:

```
rpm -qi
```

  Usage: *rpm -qi perl*

- Query rpm package for short description:

```
rpm -qi
```

  Usage: *rpm -qi perl*

- Find out what rpm package a file belongs:

```
rpm -qf <path_to_file>
```

  Usage: *rpm -qf /etc/passwd*

- Find out what rpm package a file belongs:

```
rpm -qf <path_to_file>
```

  Usage: *rpm -qf /etc/passwd*

- Find out package configuration file:

```
rpm -qc <package_name>
```

  Usage: *rpm -qc httpd*

- Display list of configuration files for a command:

```
rpm -qcf <path_to_file>
```

  Usage: *rpm -qcf /usr/X11R6/bin/xeyes*

- Display list of all recently installed RPMs:

```
rpm -qa --last
```

  Usage: *rpm -qa –last*

- Find out what dependencies a rpm file has:

```
rpm -qR <package_name>
```

  Usage: *rpm -qR mediawiki-1.4rc1-4.i586.rpm*

CHAPTER 11

ssh

- Per-host SSH client configuration options

  You can set per-host configuration options in ~/.ssh/config by specifying Host hostname, followed by host-specific options. It is possible to set the private key and the user (among many other settings) on a per-host basis.

  Here's an example config file:

  ```
  Host abhijeet
  User admin
  IdentityFile ~/.ssh/admin.id_dsa
  ```

  or:

  ```
  Host somehost
  User dev
  HostName example.com
  IdentityFile ~/.ssh/dev.id_dsa
  ```

- Add public key to remote server:

  ```
  ssh-copy-id -i ~/.ssh/id_rsa.pub username@hostname
  ```

  Usage: Copy public key to remote server

CHAPTER 12

Indices and tables

- genindex
- modindex
- search