# [Run-levels: Create, use, modify, and master](http://magazine.redhat.com/2008/06/03/run-levels-create-use-modify-and-master/" \o "Permanent Link: Run-levels: Create, use, modify, and master)

#### by [Noah Gift](http://magazine.redhat.com/author/ngift/)

The phone rings. You pick it up, and it is the recruiter for your dream job. Your palms begin to sweat as the technical interview starts.

“You want to know the run-levels for Red Hat® Enterprise Linux® or Fedora®?” You pause, thinking. “Well, I use Linux every day, and I know single user mode is level 1…”. You stammer a bit, and say you’re drawing a blank for the rest. The recruiter thanks you in that “sorry” tone-of-voice, and hangs up the phone.

Let’s cross that question off the recruiter’s list forever. (Sorry, recruiters.) In this article, we cover how to create, use, modify, and ultimately master run-levels. Bookmark this page with your favorite bookmarking service, and rest easy about ever missing that interview question again.

Of course, there are more reasons to know about run-levels than just to pass an interview. Interacting with run-levels is quite useful once you get used to it. In this article we are going to cover the basics, and then go beyond that to create our own run-level that we write a script against.

## What’s a run-level?

A run-level is a system state that is defined by the services listed in /etc/rc.d. Typically, advanced administration of a machine is done by switching run-levels (or state) to perform tasks such as minimal resource usage, run-level 3, shutdown run-level 0, or maintenance. Maintenance is run-level 1–or single user mode, as it is commonly known.

## Unix/Linux run-level comparison chart

If you happen to work with AIX, Solaris, HP-UX, Ubuntu, Fedora, Free BSD and/or Red Hat Enterprise Linux, then it can get a bit confusing figuring out which run-level does what. Please refer to the Wikipedia reference at the bottom of this article to get more information on cross platform run-level comparisons.

## Red Hat run-level chart

Here is a chart of Red Hat-specific run-levels:

|  |  |  |
| --- | --- | --- |
| **LEVEL** | **NAME** | **DESCRIPTION** |
| 0 | Halt | Immediately shuts down system and powers it off, if it can |
| 1 | Single user | Brings system to a bare essentials mode for maintenance |
| 2 | User-defined | Custom |
| 3 | Multi-user with console only | All services are running but X11 |
| 4 | User-defined | Custom |
| 5 | Multi-user with display and console | All services are running including X11 (or GUI) |
| 6 | Reboot | Reboots the machine |

## What run-level am I?

Just like life, with an operating system you need to know where you are now, in order to get to where you want to go next. If you are planning on modifying your run-level, you need to first know what your current run-level is. In order to do this, you can use one of two commands, like so:

[root@localhost ~]# who -r

run-level 3 2008-04-29 08:17 last=5

[root@localhost ~]# runlevel

5 3

If we look at the output of who -r, we can tell that we are currently running at run-level 3–which is multi-user, but console only.We can also tell that we were previously running at run-level 5, which is multi-user with console and X11 login.

## Changing run-levels

Once you know what run-level you are at, it is very simple to change to a different one. All you need to do is type: “init” followed by number of the runlevel you would like to switch to. Here is an example of switching to single user mode, or runlevel 1:

init 1

This command will change your system to single user mode, and it will ask you for the root password. When you arrive in single user mode, there are no services running, as this level it is most often used for maintenance, backup, or recovery. Once you are in single user mode it is quite common to enable, for example, network and NFS to backup your operating system, like so:

service network start; service nfs start

When you are done with your work, type in the run-level you would like to go to–perhaps run-level 5 which brings up the X11 login window:

init 5

Later in this article, we will write our own run-level and then use it to script a maintenance operation.

## Permanently changing the default run-level

While changing the runlevel manually is most common, sometimes it is useful to change the default run-level from level 5 to level 3 permanently. This can help conserve resources inside of, for example, a virtual machine. You may also choose to define your own custom run-level, and wish to make that the default.

You will need to edit /etc/inittab and change this line with your favorite text editor:

id:5:initdefault:

Change ’5′ to the run-level you wish your machine to be at when it boots. To change the run-level so that it never loads the GUI on boot would look like this:

id:3:initdefault:

Note:

A word of caution on editing /etc/inittab. It is very important to keep /etc/inittab in version control, and/or keep a backup of it when you are editing the file. If you make a change incorrectly you can render your operating system unbootable.

Tip:

If you happen to get yourself in this pickle, there is a way out. You can interrupt the Grub boot loader and press “A”, and then append the word “emergency” to the end of the kernel arguments. This will boot the operating system without using init. Then, you can fix what you altered by copying back the original version of /etc/inittab.

## Creating your own run-level HACK

First, a word of caution. Do not do this on a production machine, period! This section is a VERY dirty hack that you should only use on a virtual machine you can experiment with, or a machine you don’t mind rebuilding.. It is always a good idea to do testing inside of a virtual machine before doing something that could potentially render a box unbootable. This is a very quick and dirty way to alter a run-level for the purposes of learning, but perhaps you can get some ideas from it that can be used in a more production-oriented way. Ideally, some of the readers of this article will post some production quality hacks to creating custom run levels.

1. cd to /etc/rc.d/rc4.d/
2. do a sanity check to make sure you are running Red Hat: cat /etc/redhat-release
3. backup existing run-level directory:
4. mkdir /tmp/rc4.d.original/
5. cp /etc/rc.d/rc4.d/\* /tmp/rc4.d.original/
6. rm -f /etc/rc.d/rc4.d/\*

At this point

cp /etc/rc.d/rc1.d/\* /etc/rc.d/rc4.d/

We have now copied the run-level scripts for single user mode into our own custom run-level 4. We can hijack the the S99single script and tell it to do something different. In this example, we are going to write a custom Python script that gets forked to the background and backs up the machine over rsync. Let’s edit that file we copied:

vim /etc/rc.d/rc4.d/S99single

Change the last part of it to look like this:

# Now go to the single user level.

echo $"Telling INIT to go to single user mode."

echo "This is a custom code. Forking custom script"

/custom.py &

exec init -t1 S

We’ve inserted two lines. One echoes that we are forking off a custom script. The second line forks a python script, shown below, that backs up the machine via rsync. Note that this assumes you have set up ssh keys on the remote backup server.

custom.py script:

#!/usr/bin/env python

import time

import subprocess

rsync = "rsync -av / 10.0.1.3:/Volumes/Backup/server\_backup/"

network = "service network start"

init = "init 3"

cmds = [rsync, network]

def single\_user\_backup():

"""Starts network service, creates backup and returns to init 3"""

try:

subprocess.call(network, shell=True)

subprocess.call(rsync, shell=True)

finally:

subprocess.call(init, shell=True)

def main():

"""Runs program"""

print "sleeping for 60 seconds"

#time.sleep(60) #Gives machine time to quiesce

single\_user\_backup()

main()

The main function runs a sleep command for 60 seconds, just to give the single user mode scripts time to quiesce the box. Remember, this script is forked to the background. Next, function single\_user\_backup attempts to start network services and run rsync to remotely back up the whole / volume to another server. This is obviously crude and there will be lots of errors trying to back up /proc, for example, but it give you an idea of how an automated backup could work with a custom run-level. Finally, the machine gets called back to init 3, which is console only multi-user mode.

Tip:

Again, this is just an idea for a backup script, but not one I would actually run in production in my wildest dreams. One problem with this technique is that because of symbolic links in run level 1, we actually, changed run level 1 and our run level 4. This is not acceptable, obviously, for any sane user, but it is acceptable as a way to have fun with a disposable virtual machine!

If you can think of a more realistic backup script that would work from a custom run-level, I would love to see it. Create a how to on your blog, and then post a response to this article. Also, it would interesting to see other things such as database backups and migration done with custom run-levels as well. Leave a comment and let me know what you’d do.

## Running your own run-level

To run the newly created run-level, you only need to type:

init 4

You will then see the custom print statements we inserted. The machine  
will sleep for 60 seconds, and then run the rsync backup.

## Summary

This article covered quite a bit of ground in a short while. We went over what a run-level was, how to tell what run-level you are at, how to change run-levels, and, finally, how to make your own run-level with custom, frankenstein quality, code. Hopefully, this showed you some new tricks and spurs some ideas for further innovation with run-levels.

1. /usr/sbin/ntsysv:

Usage:

1. ntsysv - manages services in the current run-level

2. ntsysv 35 - manages services for run-levels 3 & 5

Note: ntsysv nor chkconfig starts|stops services

2. Chkconfig

Usage:

1. chkconfig --list ntpd - returns run-level environment for 'ntpd'

Note: items listed as 'off' have K (kill) scripts Note: items listed as 'on' have S (start) scripts

2. chkconfig --level 3 ntpd off - creates a K(kill) script in run-level 3

3. chkconfig --level 35 ntpd off

4. chkconfig ntpd on - enables 'ntpd' in levels 2-5

5. chkconfig ntpd off - disables 'ntpd' in levels 0-6

Note: Use 'chkconfig' from the shell or a script

Note: Use 'ntsysv' from the shell in interactive mode

Note: When controlling services using 'chkconfig', reference the name of the service as it's specified in: /etc/init.d

http://www.scribd.com/doc/61274223/Linux-Interview-Questions