Building a Raspberry Pi Office Network Server

By Dave Campbell

Keywords: Raspberry Pi, RPi, small business, office, server, network, Kerberos login, domain, DNS server, DHCP server, Samba server, Active Directory, file and printer sharing, Linux, Raspian, open source.

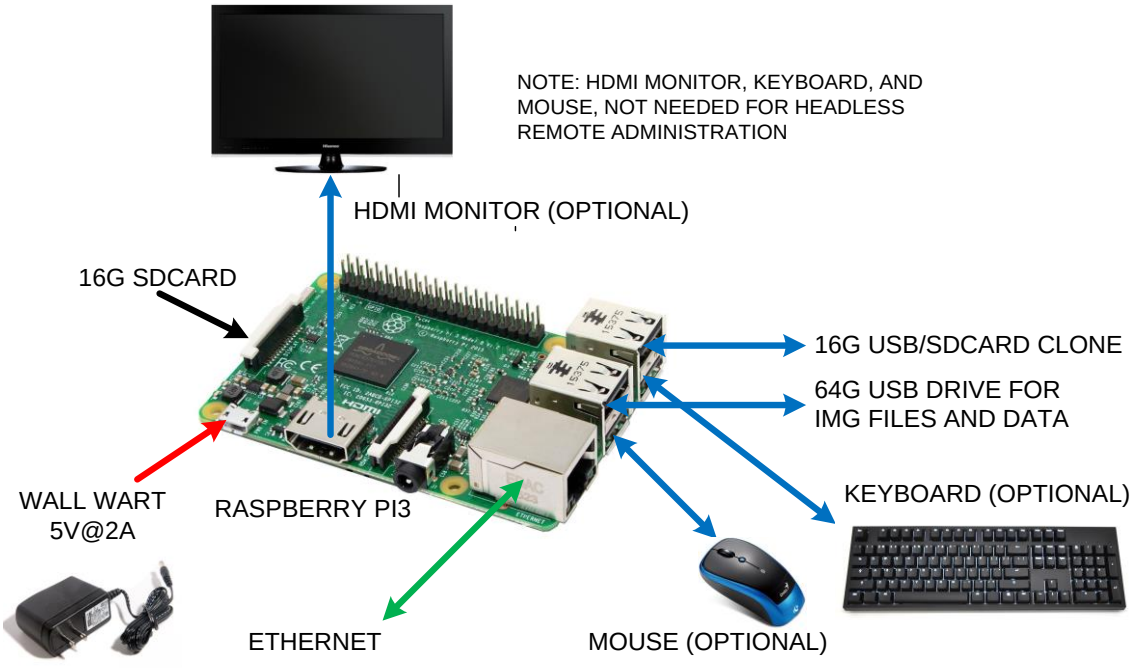

Figure 1: Raspberry Pi Server Hardware

Abstract

This document provides step-by-step details for setting up a Raspberry Pi (RPi) to perform the basic services of managing a small office or home network using Raspian Linux and open source tool packages. Most RPi applications focus on a single task, such as functioning as a wireless access point, but this setup implements a full suite of services. It includes everything needed to support a basic Active Directory domain controller, including time services, dynamic IP, DNS and DHCP network services, (file and printer sharing), Kerberos login, backup, and automation of operational tasks.

Total cost of full feature (headless) hardware, including a 1.2GHz Quad-core Raspberry Pi 3 with 1G RAM, 16G SD Card, case, power adapter, 64G USB flash drive, USB SD Card adapter and second 16G SD Card (clone), runs around $100. Total software cost and licensing is $0.

Setup time following these instructions runs 2-3 days for someone with basic Linux experience. After creating a baseline image file, it only requires a matter of a few minutes to rebuild the server. Existing servers have demonstrated uptime and reliability equal to or better than Windows based servers.

Hardware

This tutorial assumes the following hardware for setup, shown in Figure 1:

- A Raspberry Pi B+ or 2B or 3 (recommended).
- 2A MicroUSB wall wart power supply. (Note: Under powering may lead to problems.)
- 16G or larger MicroSD card loaded with NOOBS. (A Raspian only distribution optional.)
- Network connection (Ethernet recommended or wireless adapter).
- A 64G or larger USB flash drive for local Linux file and image backup.
- Optional HDMI monitor, keyboard, and mouse (required for setup only).

Operating System Assumptions

This tutorial assumes a general familiarity with Linux or at least some commandline based operating system experience such as DOS. *Appendix A: Linux Jumpstart*Appendix *A:* provides information on getting started with Linux as well as an explanation of most commands used herein.

Tips, Warnings, and Info

This tutorial shows general information notes in blue text outlined in blue boxes.

This tutorial shows tips formatted as green text outlined in green boxes.

This tutorial shows warnings formatted as red text outlined in red boxes.

Keywords

Keywords appear highlighted inline as ***example*** specific values or ***marked*** items.

Code

```
Code appears as monospaced indented brown colored text...
```

Server Specific Configuration

For consistency and simplicity of documentation, this tutorial assumes a number of configuration details specific to the tutorial that each particular setup must uniquely define. These fields, styled as shown below, include:

- Server (aliases): ***tiny*** (host), ***timex*** (NTP), ***home, house***
- User: ***pi***
- Subnet and mask: ***192.168.0.0*** or ***192.168.0.1*** / ***255.255.254.0***
- IP address(es): ***192.168.0.251*** (local DNS [DNSMasq], house)
  192.168.0.253 (Samba AD DC, home, tiny)
- Registered domain: ***saranamabq.net*** *(Required only for AD setup).*
- AD domain (& DNS) name: ***home.saranamabq.net***
- AD Kerberos Realm: ***HOME.SARANAMABQ.NET***
- NetBIOS domain: ***HOME***
- Primary DC (host): ***tiny.home.saranamabq.net (tiny)***
- SMTP admin contact: ***contact-email***
- Domain Admin Account: ***HOME\Administrator***
- Windows admin node: ***rv*** (running RSAT)

READ FIRST: I compiled this document from notes collected during early Raspian versions dating back to Woody and possibly prior. I largely wrote it based on the Jesse release. I have made revisions for Stretch but one should understand that future releases might alter instructions for a specific operation.

Given the similarity to various other Debian derivatives, most Debian instructions will work for Raspian configuration. Likewise, instructions here will largely work for other configurations, such as setup of Ubuntu on an old desktop PC.

In the event a setup task does not work as given herein, I recommend a Google search specifying Raspberry Pi, the specific Raspian version, and task.

I recommend you read the first sections of this document regarding setup and installation completely before starting.

PS. If you have not already discovered, Raspian releases, derived from the Debian Linux core, follow the Debian naming convention based on characters from Pixar's Toy Story. For more info, see https://en.wikipedia.org/wiki/Debian_version_history.

Starting with Raspian Linux

This setup assumes the popular Raspian Linux distribution, a Debian Linux derivative specifically optimized for the Raspberry Pi platform resources. The configuration instructions generally follow an order of precedence required for proper changes. That is, you must do certain things before you can complete other actions. Although the order of most things typically does not matter some things may be more easily accomplished based on certain services being available.

The following configuration setup and changes make use of the ***nano*** editor included in the default Raspian configuration, but any text editor of user choice may be used. I suggest the user spend some time familiarizing oneself with a basic text editor of choice before attempting server setup. It is always a good idea to work with recovery in mind, so I suggest reviewing the *Appendix A: Linux Jumpstart*Appendix *A:* section on *Best Practices* before making any edits to configuration setups.

The code examples provided can generally be cut and pasted directly to the Raspberry Pi commandline or into specified files as needed although some edits would be required for user and server specific information as given above.

You can install from several different Raspian images choices, as well as alternatives, all available from the raspberrypi.org download site. This document covers Raspian via NOOBS.

- **Raspian.** Basic desktop Raspian image.
- **Raspian Lite.** A lighter version more appropriate for servers or smaller dedicated installs.
- **NOOBS.** An OS installer for novice installs that lets you select the particular OS and automatically set up the SD card appropriately. Detailed below.
- **Third Party.** A number of third party images support alternative OS installs, including Ubuntu Mate, Snappy Ubuntu Core, Windows 10 IOT Core, OSMC, Libreelec, Pinit, and RISC OS.
- **Raspian Desktop.** A little misleading, Raspian desktop emulates a Raspberry Pi on a PC or Mac and is not actually a Raspberry Pi install.

Note: NOOBS and Raspian have different disk partitions that may affect other disk operations.

New Out Of Box Software (NOOBS)

NOOBS implements an operating system (OS) installer and provides a very convenient way to start building a server. Download the NOOBS installer ISO image from the Raspberry Pi web site (http://www.raspberrypi.org) and prepare an SD card according to instructions there. Once ready, simply insert the SD card and apply power to boot the Raspberry Pi to the NOOBS installer.

> ***Tip: For the easiest install, you can purchase an SD card with NOOBS preinstalled from most places that supply Raspberry Pi boards and kits.***

When booted NOOBS will load a simple graphical menu to choose one of several OSes for install. Select Raspian and install. When installation completes the ***raspi-config*** program should automatically start. See the raspi-config *Utility* section below for more information.

Basic Machine Setup

To be useful as a server with remote access you must change a few basic settings. **You may perform all these actions from a terminal window started from an icon on the Desktop, usually as a *superuser*.**

> ***Tip: The*** sudo ***command prefix used in the following instructions stands for "superuser do", explained in Appendix A: Linux Jumpstart.*** sudo ***gives you temporary permission to act as an administrator to perform privileged operating system actions. Failure to include sudo will likely result in the familiar "permission denied" message. When performing many successive commands, such as initial setup, you will find it easier to use the*** su ***command, which enters a continuous superuser mode. Type*** exit ***to return to user mode.***

Machine Reboot and Shutdown

Many operations require rebooting the system afterwards. Simply run:

```
sudo reboot
```

Alternate ways to shut down the Raspberry Pi include any of the following:

```
sudo halt -p
sudo shutdown -h now
sudo poweroff
```

These commands signal all processes to save state as necessary and stop, unmounts file systems, and then sets the RUNLEVEL=6, which stops execution and makes removing power safe.

> ***Warning: Removing power (from the Raspberry Pi while operating) to initiate a reboot has a tendency to corrupt the SD card. Use the shutdown methods outlined above. Wait for all lights to stop flashing before removing power.***

raspi-config Utility

After login, run the ***raspi-config*** utility using the commandline below and make appropriate changes to password, hostname, locale settings, and advanced options as desired using:

```
sudo raspi-config
```

> ***Note: The instructions below duplicate many of these steps but raspi-config provides a convenient one-stop utility for making a number of changes at one time.***

Upon ***reboot***, you will be presented a terminal session requesting login credentials. Login as user ***pi*** with the ***password*** (default raspberry) defined above in the ***raspi-config*** session. I recommend the following ***raspi-config*** changes:

- Change user pi Password.
  Document it! If you forget it, you will have to start over.
  See warning below. See password tip below.
- Change the hostname.
- Make the appropriate (US) localization changes. (See warning below.)
- Enable SSH under Interfacing Options to enable remote access.
  (See IMPORTANT NOTE ABOUT KEY GENERATION in Remote Secure Shell (SSH) Operation section.)
- Run Update to get the latest version of raspi-config.
- Under Advanced options:
  - Expand the file system under Advanced Options | Expand Filesystem.
    (Should happen automatically with newer versions.)
  - For headless setups (e.g. servers not used for graphics applications), minimize the graphics memory under Advanced Options | Memory Split.
  - Set Overscan and Resolution as necessary for particular display performance.

Set Root Password

Always set the root password to a known value for administrator use by running the command.

```
sudo passwd root
```

You must successfully enter the password twice to change it. It may be set the same as user ***pi*** since ***pi*** has administrator privileges. Document it!

WARNING: I recommend setting localization options and rebooting BEFORE changing the password. A keyboard change may make it impossible to reenter your password as special characters in particular may be located on different keys after the localization changes.

Tip: By best practice, an administrator password must meet rules for strict passwords. (Technically, the OS does not enforce this when setting the password, but not doing so will likely result in problems later on.) A strict password must include at least one each of upper and lower case letters, digit 0-9, and special characters (i.e. ~!@#$%^&*_-+=`\|\(){}[]:;"'<>,.?/), be at least 8 characters in length, and it must not contain the username.

Info: The majority of hacking breaches STILL result from default and weak passwords.

Hostname

The human-friendly machine name, known as the hostname, can be edited using ***raspi-config*** or by directly editing the ***/etc/hostname*** file with ***nano***. This file contains a single line containing the name of the machine.

```
sudo nano /etc/hostname
(Enter the desired machine name without a newline)
(Save the file by CTRL-X)
```

For more than one RPi, common practice involves picking a naming theme, for example, colors.

Host Lookups

The */etc/hosts* file relates fixed IP addresses of local machines to their static host names and aliases. Edit the file with

```
sudo nano /etc/hosts
```

Add a line for the server itself and a second for the primary domain controller and include all desired aliases as a space delimited list such as

```
192.168.0.251    house timex
192.168.0.253    tiny home home.saranamabq.net
```

> ***Note: See Network Time Protocol (NTP) Service section for use of*** timex***.***

Network Interface Configuration

Prior to the *"Jesse"* release of Raspian, edit the */etc/network/interfaces* file to change the configuration from DHCP to a fixed IP address. (See

DNS & DHCP Services section for further understanding.) While a server can function with a DHCP address, it will lead to more problems than it is worth. From a terminal window type

```
sudo nano /etc/network/interfaces
```

Change any of the lines

```
iface eth0 inet dhcp
iface eth0 inet manual
```

to

```
#iface eth0 inet dhcp
#iface eth0 inet manual
```

The **#** at the beginning of the line disables the configuration of that line. Then add the following lines (in order) after it

```
auto eth0
iface eth0 inet static
      address 192.168.0.251
      gateway 192.168.0.1
      netmask 255.255.254.0
      broadcast 192.168.0.255
      network 192.168.0.0

auto eth0:0
iface eth0:0 inet static
      address 192.168.0.253
```

Note: The alias** eth0:0 **definition creates an alternate address for the same interface to support Samba configuration notions.

For *"Jesse"* **and later Raspian versions** edit the /etc/dhcpcd.conf file per https://www.modmypi.com/blog/how-to-give-your-raspberry-pi-a-static-ip-address-update

Reboot for the changes to take place.

To verify the configuration run the ***ip*** command. It should identify to IP addresses for eth0.

```
ip addr
```

Tip: 192.168.0.0 is the most common local subnet but you may use other addresses reserved for local use consistent with other hardware setups of the local network.

Tip: Subnets may be joined into a single net by the appropriate configuration of the netmask. For example, a netmask of 255.255.254.0 will combine 192.168.0.0 and 192.168.0.1 subnets.

Tip: The IP address must be unique per machine. Recommend numbering servers from the top down of the local subnet. Address 255 represents a reserved broadcast address making the highest possible server address 254. The gateway represents the box that connects the local network to the next network, likely your cable or DSL modem, typically 192.168.0.1.

Network Time Protocol (NTP) Service

Secure access services use "current time" as a component, so machines not synchronized to a time standard will have difficulty connecting or staying connected. The network time protocol (NTP) service synchronizes a machine's clock to international standard time. By default, Raspian automatically starts

the NTP daemon (ntpd) and configures it to acquire time randomly from a pool of calibrated timeservers.

The timex Alias

By default, the server will operate as a local time server. I recommend configuring the server in the ***/etc/hosts*** file to have an alias of ***timex*** or similar. All other local machines can then sync to ***timex*** reducing traffic overhead for timeservers. If in the future you replace the server, giving the new server the same ***timex*** alias will allow all the network machines to continue to sync, saving you the time and effort to reconfigure them.

Update Operating System

Before doing much else run the following command string. The first part (before &&) updates the local copy of the code repository listing and the second part (after &&) then upgrades the OS based on this listing if the first is successful. The -y switches answer yes to questions to suppress interactive queries and automatically run. You can run each part independently, without &&.

```
sudo apt-get update –y –fix-missing && sudo apt-get upgrade -y
```

After updating the OS this first time, I recommend rebooting. Simply run:

```
sudo reboot
```

Following the reboot, log in and start a new terminal to continue.

> ***Tip: See Automatic Server Update section for information on automating updates.***

Prior to Jesse, if setting up on a **Raspberry Pi B+** for a **Raspberry Pi 2** or **Raspberry Pi 3**, then also run the following. This way the SD card will work on both platforms interchangeably.

```
sudo apt-get dist-upgrade
sudo apt-get install raspberrypi-ui-mods
```

X Windows Graphical User Interface

Raspian supports the MIT X Windows system as a graphical user interface (GUI). After login to a terminal window, start the Raspberry Pi graphical mode if desired simply by running the command:

```
startx
```

Remote Secure Shell (SSH) Operation

At this point the server is ready for remote operation and all remaining setup can be done remotely either from another machine on the local network or from an off-site Internet location using Secure Shell (SSH). From another Linux box on the local network, use the following command (or a Windows machine using Putty, see *Certificate* Based Login

You can use SSH to make machine-to-machine connections too such as one machine backing up another. In this case, login uses a security certificate. It involves using openssl ***key-gen*** to generate a *self-signed certificate* and then placing that certificate on both machines. See ***man ssh*** for more instructions. Reference gives some examples of setup and use of certificate based login.

Port Forwarding

In order to use Secure Shell over the Internet, the DSL/cable modem box at the server must enable port forwarding of the WAN port 22 to the LAN SSH server port 22.

NOTE: Due to the significant number of modem/router boxes on the market, defining the instructions for port forwarding exceeds the scope of this document. See port forwarding instructions for your specific model modem. Google it!

PuTTY section below).

```
ssh pi@tiny      or    ssh pi@192.168.0.251
```

This provides a terminal window on the client machine connected remotely to the server as user ***pi***. If the local network does not recognize the machine name, you may need to use the specified IP address as shown in the alternate command form. (This may be necessary until DNS is established later.) After Internet connection setup the ***ssh*** command can provide access from anywhere using the fully qualified domain name.

You can perform all further configuration via SSH without having to be present at the machine. SSH becomes particularly handy when performing maintenance from an off-site location such as home. (Access from the Internet will require some additional setup, discussed later, depending on your Internet Service Provider (ISP).)

SSH access also enables headless operation, meaning a machine operating without a monitor, which saves on server cost.

IMPORTANT NOTE ABOUT KEY GENERATION

When setting up a new SSH installation, especially from a downloaded image created for a standalone application, you should always run ssh-keygen to create unique keys for the specific installation. For example:

```
sudo ssh-keygen –t rsa –f /etc/ssh/ssh_host_rsa_key
```

This will create both /etc/ssh/ssh_host_rsa_key and /etc/ssh/ssh_host_rsa_key.pub

The next login after this change may result in an error. Fix this by deleting the ./.ssh/known_hosts file on the local machine.

To verify the fingerprint of the remote machine run the command…

```
ssh-keyscan tiny >/tmp/key; ssh-keygen -lf /tmp/key; rm /tmp/key
```

Certificate Based Login

You can use SSH to make machine-to-machine connections too such as one machine backing up another. In this case, login uses a security certificate. It involves using openssl ***key-gen*** to generate a *self-signed certificate* and then placing that certificate on both machines. See ***man ssh*** for more instructions. Reference [1] gives some examples of setup and use of certificate based login.

Port Forwarding

In order to use Secure Shell over the Internet, the DSL/cable modem box at the server must enable port forwarding of the WAN port 22 to the LAN SSH server port 22.

NOTE: Due to the significant number of modem/router boxes on the market, defining the instructions for port forwarding exceeds the scope of this document. See port forwarding instructions for your specific model modem. Google it!

PuTTY

If working from an MS Windows or Mac machine the ***PuTTY*** terminal emulator provides a nice graphical user interface (GUI) based client for connecting to the server. Simply install it on a Windows machine and define a configuration to point to your server. PuTTY will work for both local network connections as well as access from the Internet.

Tunneling

Secure Shell implements an encrypted link between the two connected machines that means a significant security advantage for connecting across the Internet. This capability extends to any other application based on a procedure called tunneling. ***Putty*** provides a convenient way to setup and use SSH for any remote application. By tunneling other applications, all communications occur through a single port (22) without the need to open other ports and increase the server's vulnerability.

> ***Tip: Windows Remote Desktop, VNC, and any other application can run via a SSH tunnel. As such, your router only requires forwarding of port 22, rather than opening a unique port for each application.***

Terminal Multiplexer (tmux)

SSH sessions interact with an active terminal window. When SSH disconnects that terminal session stops and any current action such as an executing script will terminate and likely fail. This can be frustrating, for example if you begin compiling something that takes several hours only to have you client machine go to sleep or lose communications and terminate the session before completion.

The ***terminal multiplexer*** (***tmux***) program provides an excellent workaround for this. It connects your SSH session to a terminal client/server on the host that stays running even if the connection to the host ends. Additionally, it supports multiple windows and panes from a single SSH session.

Server operation does not require ***tmux***, but it greatly helps with managing it. *0/usr/local/bin/update*

Script to automatically make server OS updates and notify admin of action, errors, and pending reboot. Note: Accesses /etc/motd (message of the day shown at login) to query reboot.

```
#!/bin/bash
# script to make OS updates

e=`date +%s`
d=`date`
logRoot='/tmp/update'
tmp="$logRoot.tmp"
log="$logRoot.$e"
host=`cat /etc/hostname`

echo "$0 RUN $d with $PATH" > $log 2>&1
echo "" >> $log 2>&1
# run system update...
apt-get update -y > $tmp 2>&1
apt-get upgrade -y >> $tmp 2>&1
echo "" >> $tmp 2>&1
echo "" >> $tmp 2>&1

# remove logs over 30 days old...
find $logRoot.* -mtime +30 | 2>/dev/null xargs -r rm -- >> $tmp 2>&1
echo "" >> $tmp 2>&1
```

```
# is restart required...
msg=$(grep "restart required" /etc/motd)
check=$(grep -c "restart required" /etc/motd)
echo "MSG: $msg" >> $log 2>&1
if [ $check -ne 0 ]
then
  echo "REBOOT" >> $log 2>&1
else
  echo "OK" >> $log 2>&1
fi
echo "" >> $log 2>&1

echo "Errors?..." >> $log 2>&1
grep -i "error" $tmp >> $log 2>&1
echo "-- END OF ERRORS --" >> $log 2>&1
echo "" >> $log 2>&1

echo "Update Log..." >> $log 2>&1
cat $tmp >> $log
cp $log "$logRoot.log"
cat $log | /usr/local/bin/mailto.py "$host update script..."
```

/usr/local/bin/ups

Script called from apcupsd daemon to notify admin of UPS state changes such as lost power. Links must be added to /etc/apcupsd folder to point to this script for each desired power state warning. For example

```
sudo ln –s /usr/local/bin/ups /etc/apcupsd/doshutdown
```

This causes an ***apcupsd doshutdown*** event to call ***/usr/local/bin/ups***. Other states include ***mainsback***, ***offbattery***, ***onbattery***, ***powerout***, etc. See ***man apcupsd*** for details.

```
#!/bin/bash

# script to report UPS state changes...

e=`date +%s`
d=`date`
logRoot='/var/log/ups'
log="$logRoot.$e"
host=`cat /etc/hostname`

echo "$0 $* RUN $d" > $log 2>&1
echo "" >> $log 2>&1

# dump apcupsd log...
tail /var/log/apcupsd.events >> $log 2>&1
echo "" >> $log 2>&1

# remove logs over 30 days old...
find $logRoot.* -mtime +30 | 2>/dev/null xargs -r rm -- >> $log 2>&1

# move log to base file and email to sysop...
cp $log "$logRoot.log"
cat $log | /usr/local/bin/mailto.py "$host UPS script..."
```

/etc/init.d/vncserver

Script to control operation of tightvncserver.

```
#!/bin/sh
### BEGIN INIT INFO
```

```
# Provides: vncboot
# Required-Start: $remote_fs $syslog
# Required-Stop: $remote_fs $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Start VNC Server at boot time
# Description: Start VNC Server at boot time.
### END INIT INFO

USER=root
HOME=/root

export USER HOME

case "$1" in
 start)
   echo "Starting VNC Server"
   #Insert your favoured settings for a VNC session
   /usr/bin/vncserver :0 -geometry 1280x800 -depth 16 -pixelformat rgb565
   ;;

 stop)
   echo "Stopping VNC Server"
   /usr/bin/vncserver -kill :0
   ;;

 *)
   echo "Usage: /etc/init.d/vncboot {start|stop}"
   exit 1
   ;;
esac

exit 0
```

Terminal Multiplexer (tmux), Install, Setup, and Use provides install, setup, and useful customization instructions.

Virtual Network Computing (VNC) Server

A virtual network computing or VNC server provides a means of seeing a desktop remotely, thus it offers a graphical or GUI alternative to secure shell. (VNC on a Linux box is essentially equivalent to the MS Windows Remote Desktop and Remote Assistance.) To setup VNC perform the following command.

```
sudo apt-get install tightvncserver
```

Start the TightVNC server on demand simply by running

```
tightvncserver &
```

The ***&*** character at the end starts the server running in the background and directly returns control to the local shell.

Or ***tightvncserver*** can be configured to always start when the machine boots. First, create a script to start and stop the server

```
sudo nano /etc/init.d/vncserver
```

(Copy the contents of the /etc/init.d/vncserver script from 0

See also Appendix C: Custom Scripts for useful “embellished” commands, such as pid, strip, update.

Appendix A: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER***.***

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

```
Custom Scripts)
(Save the file)
sudo chmod 755 /etc/init.d/vncserver
sudo update-rc.d vncserver defaults
```

To use VNC you must also install the TightVNC viewer on the client machine.

> ***WARNING: For early release of Stretch, some users reported problems running VNC under certain conditions.***

Tunneling VNC

If using VNC across the open Internet, set it up with SSH tunneling. To do so using Putty, simply enter "***L5900 black:590x***" into the ***Connection|SSH|Tunnels*** list of forwarded ports, where ***x*** is the number of the server started by the ***tightvncserver*** command.

DNS & DHCP Services

Domain naming service (DNS) and dynamic host configuration protocol (DHCP) address assignments implement fundamental services of a working network domain. A server must provide these functions on the local area network (LAN).

DNS resolves the *easily remembered human-form machine names*, such as ***tiny***, into their respective *numerical internet protocol addresses* (IP) like ***192.168.0.253***. The DNS server provides this service for the local network and queries the next level domain service for this information for addresses outside the local network, such as resolving ***google.com***.

Machines may have fixed IP addresses or dynamically assigned addresses, but every machine on the local network must have a unique address. The DHCP service does the work of dynamically assigning addresses. Machines configured for DHCP request an address from an available network DHCP server upon power up. The DHCP server allocates addresses to requests and catalogs who's who by name and address for use by DNS.

> ***Tip: Running a local DNS server will speed up Internet access and resolve local machine names without burdening the ISP. (Note: without a local server, requests for local machines are forwarded to the ISP, which likely does not know anything about your machines, and timeout after 2 seconds.)***
>
> ***Tip: I recommend configuring servers with fixed addresses and other machines dynamically for minimal problems.***

> ***Warning: If running a DHCP service as part of the server, be sure to disable any DHCP service provided by your DSL or cable modems to prevent conflicts.***

Network Utilities

By default, Raspian does not include certain utilities useful to DNS. You may install them using

```
sudo apt-get install dnsutils
```

This adds ***nsupdate***, ***dig*** and ***nslookup*** functions. These utilities provides helpful diagnostics for network problems. For example, ***nslookup timex*** may report…

```
Server:         192.168.0.253
Address:        192.168.0.253#53

Name:   timex.saranam
Address: 192.168.0.251
Name:   timex.saranam
Address: 192.168.0.250
```

This indicates that two machines on the local domain both map to ***timex***. Either machine can respond to requests to ***timex***. The ***dig*** command gives a bit different diagnostic information but the same IP to name mapping. For example, ***dig black*** reports.

```
; <<>> DiG 9.8.4-rpz2+rl005.12-P1 <<>> black
;; global options: +cmd
;; Got answer:
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 1198
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;black.                         IN      A

;; ANSWER SECTION:
black.                  0       IN      A       192.168.0.250

;; Query time: 7 msec
;; SERVER: 192.168.0.250#53(192.168.0.250)
;; WHEN: Tue Aug 11 19:41:43 2015
;; MSG SIZE  rcvd: 39
```

Note the only line that does not begin with a semicolon gives the mapping of name to IP address. The response also returns the time it took to resolve the answer useful for diagnosing network delays.

Ping

Another useful network utility already installed with Raspian is ***ping***. The receiving machine echos the ping request in the same sense sonar bounces off a ship. For example ***ping –c 5 tiny*** may reply…

```
PING tiny (192.168.0.253) 56(84) bytes of data.
64 bytes from tiny (192.168.0.253): icmp_req=1 ttl=64 time=0.649 ms
64 bytes from tiny (192.168.0.253): icmp_req=2 ttl=64 time=0.643 ms

--- tiny ping statistics ---
5 packets transmitted, 2 received, 60% packet loss, time 4005ms
rtt min/avg/max/mdev = 0.643/0.646/0.649/0.003 ms
```

This sends 5 pings (i.e. -c) to host ***tiny*** and reports the results. In this case only 2 echos indicates a poor connection — a significant amount of data loss. Packets that complete echo in under 1 ms.

> ***Note: Not all machines or devices support or honor ping and may not reply at all even with no problem present.***

(NoIP) Dynamic Update Client

Network servers generally have a wide area network (WAN) presence, meaning they connect to the Internet. Just as machines on the LAN require an IP address, the server must have an IP address on the WAN to communicate with other machines. The Internet service provider (ISP) may assign the WAN address statically or by DHCP. A dynamically assigned address may change any time.

This server likely has a registered Internet domain name as well, such as ***saranam.net***. This requires a function to synchronize the server's domain name to its WAN-IP address as it changes, referred to as a dynamic update client (DUC). The following defines the setup for the No-IP dynamic update client, which assumes the use of the No-IP dynamic addressing hosting service. (For first time setup, first create an account on ***no-ip.com*** and define the host.

Download the latest client (assumes 2.1.9) from No-IP to a temporary folder, extract, build, install, and run to create a configuration file...

```
mkdir /tmp/noip
cd /tmp/noip
wget http://www.no-ip.com/client/linux/noip-duc-linux.tar.gz
tar vzxf noip-duc-linux.tar.gz
```

```
cd noip-2.1.9-1
sudo make
sudo make install
sudo /usr/local/bin/noip2 -C
```

Running the client will ask for information, including the No-IP login username and password, on how to update clients. After this setup, rerun the client without the –C option to start it in the background.

Perform the following to make the DUC run each time the Raspberry Pi reboots.

```
sudo nano /etc/init.d/noip2

(Copy the contents of the NoIP DUC script from 0

See also Appendix C: Custom Scripts for useful “embellished” commands, such as pid,
strip, update.
```

Appendix B: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER.

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

```
Custom Scripts and save)
```

Make the script executable and set it to run on startup.

```
sudo chmod 755 /etc/init.d/noip2
sudo update-rc.d noip2 defaults
```

To start stop or restart the service at any time do…

```
sudo /etc/init.d/noip2 start
sudo /etc/init.d/noip2 stop
sudo /etc/init.d/noip2 restart
sudo /etc/init.d/noip2 status
```

To update the configuration at any time, first stop the service then run

```
/usr/local/bin/noip2 -C
```

To stop the DUC service permanently, run…

```
sudo update-rc.d noip2 remove
```

Tip: Create a symbolic link as …

sudo ln -s /usr/local/bin/noip2 /usr/local/bin/noip

and refer to noip anywhere noip2 is used. If the version of noip changes to noip3, just redefine the link and continue referring to noip. This represents a common Linux practice that allows commands to point to the latest version transparent to the user.

After completing the install, reboot the Raspberry Pi with the following line and verify that the noip daemon is still running.

```
sudo reboot
```

Tip: 0

See also Appendix C: Custom Scripts for useful "embellished" commands, such as pid, strip, update.

Appendix C: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER***.***

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

> ***Custom Scripts contains a script for pid that can be run to list process IDs matching a string, such as pid noip***

DNSMasq

> ***NOTE: Running multiple DHCP servers on a local network will result in many bazaar problems, as different machines will potentially receive their setup from an arbitrary source. Routers, such as a Comcast or CenturyLink modem, often run the service by default and must be disabled before using DNSMASQ. Similarly, other Raspberry Pi devices on the network may be running DNSMASQ. To disable DNSMASQ from automatically running at startup, edit the /etc/default/dnsmasq file and set ENABLE=0.***

DNSMasq (short for DNS Masquerade) provides both a simple DNS server and DHCP server in a single setup. For installation (if necessary), simply run…

```
sudo apt-get install -y dnsmasq
```

After installation copy the default configuration to a domain configuration file and edit as in...

```
sudo cp /etc/dnsmasq.conf /etc/dnsmasq.d/saranam.conf
sudo nano /etc/dnsmasq.d/saranam.conf
```

The configuration file provides many comments for straightforward setup. Below is a specific recommended configuration for starting. Many settings simply require removing the # from the start of the line.

```
# basic domain settings...
domain-needed
bogus-priv
resolv-file=/etc/resolv.dnsmasq
local=/saranam/
expand-hosts
domain=saranam
cache-size=10000
# example of manually associating mac address, name, and IP address...
dhcp-host=00:02:2A:4A:C2:23,ojo,192.168.0.81
# this setting controls dhcp assignment range addresses 51-99
dhcp-range=192.168.0.51,192.168.0.99,12h
# these settings provide WINS server data to Windows machines
dhcp-option=44,192.168.0.253    # WINS server
dhcp-option=46,8                # WINS hybrid node type
dhcp-option=3,192.168.0.1       # gateway
dhcp-option=1,255.255.255.0     # subnet mask
dhcp-authoritative
# identify bogus and non-existent domains (e.g. ad sites)
bogus-nxdomain=63.146.68.202
bogus-nxdomain=63.146.68.201
```

Next, build the /etc/resolv.dnsmasq file referenced in the configuration

```
sudo nano /etc/resolv.dnsmasq
```

Include a (prioritized) list of servers to query for forwarded DNS requests such as…

```
# dnsmasq configured to look for forwarding servers here...
# qwest.net...
nameserver      205.171.3.25
nameserver      205.171.2.25
# google public...
nameserver      8.8.8.8
nameserver      8.8.4.4
# openDNS...
nameserver      208.67.222.222
nameserver      208.67.220.220
#gateway...
nameserver      192.168.0.1
```

Edit the default /etc/resolv.conf file as well for any local services that may make queries.

```
sudo nano /etc/resolv.conf
```

NOTE: For Jesse and later Raspian versions, edit /etc/resolvconf.conf file instead.

Paste the following into the file

```
domain saranam
search saranam
nameserver 192.168.0.253
```

Edit the /etc/hosts file to include the names and IP addresses of any static machines.

After configuration or /etc/hosts changes, restart the DNS service to have settings take effect by

```
sudo service dnsmasq restart
```

or

```
sudo /etc/init.d/dnsmasq restart
```

You can test the server by running a number of local and remote DNS requests with ***dig*** or ***nslookup*** such as

```
nslookup google.com
dig google.com
dig tiny
```

It may also be helpful to perform some web service requests using a browser or ***wget***.

Uninterruptable Power Supply Daemon (APCUPSD)

The APC uninterruptable power supply (UPS) daemon monitors the data port of an UPS as a background service and notifies the server of power outages and pending power failure. (NOTE: Since the Raspberry Pi does not have a power switch, this service does not actually remove power from the Pi, but does shutdown critical services and hardware for safe power failure and logs the outage. To install simply run

```
sudo apt-get install apcupsd
```

Start the UPS and connect to the server. Run the following to identify the device and see that it is recognized

```
lsusb
```

It will report something similar to the following:

```
Bus 001 Device 011: ID 051d:0002 American Power Conversion Uninterruptible Power Supply
```

Edit the daemon configuration to define the type of UPS and connection.

```
sudo nano /etc/apcupsd/apcupsd.conf
```

Also, edit the defaults file

```
sudo nano /etc/default/apcupsd
```

You MUST define the following line

```
ISCONFIGURED=yes
```

After configuring apcupsd start it using the service command as

```
sudo service apcupsd start
```

You can setup the daemon to send alerts for various conditions. See the /usr/local/bin/ups script in *0*

See also Appendix C: Custom Scripts for useful “embellished” commands, such as pid, strip, update.

Appendix D: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER***.***

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

Custom Scripts and man apcupsd for more info.

Automating Operations

One of the beauties of the Linux operation system involves the ability to script mundane tasks. This section addresses a number of such routine tasks. Such operations may vary significantly from setup to setup. Information presented here serves more as a skeleton than an actual prescription to provide the notions and mechanisms. Examples here represent a few of the tasks usually handled by any system. Extend as desired.

Admin Notifications

0

See also Appendix C: Custom Scripts for useful “embellished” commands, such as pid, strip, update.

Appendix E: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER.

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

Custom Scripts includes a script for automatically notifying system administrators by email of actions performed by other scripts.

Create the notification script by

```
sudo nano /usr/local/bin/mailto.py

(Copy the contents of the /usr/local/bin/mailto.py script from 0

See also Appendix C: Custom Scripts for useful "embellished" commands, such as pid,
strip, update.
```

Appendix F: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER.

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

```
Custom Scripts into the file and save.)
```

Note that this script contains a number of setup specific parameters that you must modify for your network. These include:

subject:	*The email subject line.*
you:	*The default email recipient(s).*
me:	*Identify string for script sender address.*
server:	*SMTP server name*
port:	*SMTP port*
username:	*SMTP login username*
password:	*SMTP login password*

> WARNING: Normally it is not good practice to embed passwords into scripts. In this case, it simply provides access to an email account (for sending only) so does not involve as serious a security issue if compromised as would an account login, but must still be secured. Be sure to

protect the file by setting the permissions to 750 and owner to root:adm with *chmod* and *chown* commands. This ensures that only root and admins can run and see the file contents. *See Appendix G: Linux Jumpstart* for help.

To call this script pass it a custom message subject and optional list of recipients as

```
/usr/local/bin/mailto.py "My Test Script" email-address < update.log
```

The script receives input from ***stdin*** so scripts can redirect content directly to it.

Automatic Server Update

0

See also Appendix C: Custom Scripts for useful "embellished" commands, such as pid, strip, update.

Appendix G: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER.

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

Custom Scripts includes an update script for automatically keeping the server up to date.

Create the update script by

```
sudo nano /usr/local/bin/update

(Copy the contents of the /usr/local/bin/update script from 0

See also Appendix C: Custom Scripts for useful "embellished" commands, such as pid,
strip, update.
```

Appendix H: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER***.***

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

```
Custom Scripts into the file and save.)
```

> ***Tip: Save your scripts in /usr/local/bin. The operating system preserves this area on upgrades.***

Then establish the ***cron*** tasks by

```
sudo nano /etc/cron.d/sysop
```

Add these lines

```
# OS update @ 9:35PM every Friday...
35      21      *       *       5       root    /usr/local/bin/croncall update
```

See the

Strings

Because the commandline parses by space between tokens, fields containing spaces must be quoted. Often, it does not matter whether using single quotes or doubles quotes, but as a rule, generally the shell (i.e. parser) cannot see inside single quotes but can see inside double quotes. This means variable substitutions inside strings generally require double quotes and to protect again misinterpreting content as special requires single quotes and/or escaping.

Comments

Commands prefixed with a ***#*** character represent comments for Bash shell operations. While not common for interactive use, comments commonly occur throughout scripts to document actions of the script.

> ***Tip: Other scripting languages may define a different specific comment character.***

Man Pages – Linux Help

The Linux manual exists online for access interactively while working from the commandline.

```
man cat
```

Typing ***man*** followed by a command as in this example will open the help pages for that command in the current shell terminal. The user can scroll or page up and down to move around. When finished type ***Q*** or ***q*** to quit and return to the shell. You can also access man pages online from any of a number of web sites.

Program Installation

apt-get

The apt-get command, short for aptitude get where aptitude represents the name of the code Linux code repository installer, installs or removes pre-packaged software builds. It also implements commands to update code references and upgrade already installed packages.

Cron Jobs section of *Appendix I: Linux Jumpstart* for details on how to alter these settings.

> ***Tip:*** cron ***processes all files it finds in the /etc/cron.d folder, so you can name it anything. Here sysop denotes system operations.***

With this setup, the system will automatically update every Friday at 9:35 PM and send the system administrator an email confirming the operation. The notification email displays a message indicating if the server requires a reboot, captures errors, and includes a log of the updates.

Backup

Server Configuration Files Backup

Backup of server configuration files requires a Linux formatted disk since NTFS and FAT type partitions do not preserve file permissions. This does not require a significant amount of disk space since it amounts to small portion of the SD card, but must be external to the SD card. So this instruction assumes to use of a USB flash drive inserted directly into the Raspberry Pi.

First, prepare a USB flash drive using the ***fdisk*** command in (Windows or Linux) to create a single partition on the USB drive.

Then install a Linux file system to the partition by

```
sudo mkfs.ext4 /dev/sda1 -L usb
```

Warning: You must determine the correct drive and partition. This can be done by inserting the USB drive, then perform* ls -l /dev/disk/by-id *and look for disk that matches the descriptor of the USB drive. Or look for what changes when the drive is or is not inserted.

Warning: The USB drive may automatically mount in which case it must first be unmounted before creating the file system using the command:

sudo umount /dev/sda1

Either an ***ext3*** or ***ext4*** file system may be used. In the command above ***usb*** is the drive label. The /dev/sda1 field represents the first partition of drive being defined.

Once you have created the file system, permanently mount the drive at startup by adding it to the /etc/fstab file. Edit /etc/fstab with

```
sudo nano /etc/fstab
```

Add the lines at the end

```
# configuration usb...
/dev/disk/by-label/usb /mnt/usb ext4 defaults 0 0
```

Save the file. Then create the mount point and reboot the Raspberry Pi with the following

```
sudo mkdir /mnt/usb
sudo reboot
```

Once rebooted run the mount command to list mounted drives, which should include /backup.

```
mount
```

Then create needed backup directories as

```
sudo mkdir /mnt/usb/backup
sudo mkdir /mnt/usb/backup/tiny
sudo mkdir /mnt/usb/backup/tiny/daily
sudo mkdir /mnt/usb/backup/tiny/weekly
sudo mkdir /mnt/usb/backup/tiny/monthly
```

Backup Script

The backup script reads backup instructions, referred to as list files, to perform backup of desired files at specified periodic intervals, typically daily, weekly, monthly, but easily specified as desired.

Create a backup script by

```
sudo nano /usr/local/bin/backup

(Copy the contents of the /usr/local/bin/backup script from 0

See also Appendix C: Custom Scripts for useful “embellished” commands, such as pid,
strip, update.
```

Appendix I: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER.

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

```
Custom Scripts into the file and save.)
```

Then using */usr/local/bin/backup.lst.daily as a template create the command lists used by the backup script for daily, weekly, and monthly backups as*

```
sudo nano /usr/local/bin/backup.lst.daily

(Copy the contents of the /usr/local/bin/backup.lst.daily script from 0

See also Appendix C: Custom Scripts for useful "embellished" commands, such as pid,
strip, update.
```

Appendix J: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER.

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

```
Custom Scripts into the file, edit as appropriate, and save.)

sudo nano /usr/local/bin/backup.lst.weekly

(Copy the contents of the /usr/local/bin/backup.lst.daily script from 0

See also Appendix C: Custom Scripts for useful "embellished" commands, such as pid,
strip, update.
```

Appendix K: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER.

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

```
Custom Scripts or from /usr/local/bin/backup.lst.daily into the file, edit as
appropriate, and save.)

sudo nano /usr/local/bin/backup.lst.monthly

(Copy the contents of the /usr/local/bin/backup.lst.daily script from 0

See also Appendix C: Custom Scripts for useful "embellished" commands, such as pid,
strip, update.
```

Appendix L: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER***.***

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

```
Custom Scripts or from /usr/local/bin/backup.lst.daily into the file, edit as
appropriate, and save.)
```

> ***Tip: The backup script provides for some variable substitution for simplifying porting of backup lists across machines such as the mount point. These can be altered or expanded upon in the*** /usr/local/bin/backup ***script.***

Then establish the ***cron*** tasks to automate backup by

```
sudo nano /etc/cron.d/sysop
```

Add these lines

```
# Backup script run every day @ 11:05PM...
5       23      *       *       *       root    /usr/local/bin/croncall backup daily
# Backup script run every Sunday @ 11:10PM...
10      23      *       *       0       root    /usr/local/bin/croncall backup weekly
# Backup script run every month @ 11:15PM...
15      23      1       *       *       root    /usr/local/bin/croncall backup monthly
```

See the

Strings

Because the commandline parses by space between tokens, fields containing spaces must be quoted. Often, it does not matter whether using single quotes or doubles quotes, but as a rule, generally the shell (i.e. parser) cannot see inside single quotes but can see inside double quotes. This means variable substitutions inside strings generally require double quotes and to protect again misinterpreting content as special requires single quotes and/or escaping.

Comments

Commands prefixed with a ***#*** character represent comments for Bash shell operations. While not common for interactive use, comments commonly occur throughout scripts to document actions of the script.

> ***Tip: Other scripting languages may define a different specific comment character.***

Man Pages – Linux Help

The Linux manual exists online for access interactively while working from the commandline.

```
man cat
```

Typing ***man*** followed by a command as in this example will open the help pages for that command in the current shell terminal. The user can scroll or page up and down to move around. When finished type ***Q*** or ***q*** to quit and return to the shell. You can also access man pages online from any of a number of web sites.

Program Installation

apt-get

The apt-get command, short for aptitude get where aptitude represents the name of the code Linux code repository installer, installs or removes pre-packaged software builds. It also implements commands to update code references and upgrade already installed packages.

Cron Jobs section of *Appendix M: Linux Jumpstart* for details on how to alter these settings.

Data Backup

Data backup can be accomplished multiple ways depending on the setup.

Using The Backup Script

Data backup can use the same backup script used for configuration to back up (server) user data with a few possible exceptions. If backing up Windows data, the destination drive likely needs to be a larger drive and may be better if formatted as a NTFS volume and the mount must include ACLs support to preserve Windows file permissions, best handled with a Windows box.

Windows Backup

A number of tools support Windows based backup. Robocopy is a commandline based “robust copy” tool. Richcopy provides a GUI frontend for robocopy; however, it does not support all options. These tools offer much more effective copying and will not hang or crash on failure.

Using Google Drive

If using a shared network drive as a Windows file server, it can be setup as the folder location for Google Drive to automatically sync with online storage. This provides both data backup and off-site reliable safe storage. It simply requires installation of Google Drive and configuration on the Windows machine. As noted in the Samba setup, this provides an alternative to a Samba based file sharing setup. By using a USB drive, it provides a convenient setup easily ported to new machines.

Additionally, the major Google Drive folders could be redundantly backed up to a separate USB drive or even to other normally "hidden" folders within Google Drive.

SD Card Backup

Once you have completed a baseline installation clone or copy the SD card for a backup. Do this periodically as well such as once a month or quarter to make server recovery quick and easy. The instructions below identify multiple means to duplicate an SD card.

> ***Warning: SD Cards do not have the same reliability as hard disks. Loss of power or repeated writing of the same location may corrupt an SD card. I recommend routine backup.***

Cloning

Cloning consists of making an image of the SD card, which copies all of the information from the card into a single ***image*** file making it easy to restore. Cloning is a manual process and has some specific issues, namely that the clone must have an equal or larger number of sectors as the original. It may not work in all cases. See *Appendix R: Cloning a Disk* for the process.

Backup using *rsync*

A second method to backup the SD card uses the system ***rsync*** command, essentially the same as done with the backup script but with a bit different setup. Using a mounted USB to SD card adapter, you can automate this method the same as the backup process.

1. Create a second SD card with NOOBS installed as described in section *New Out Of Box Software (NOOBS)*New Out Of Box Software (NOOBS) This card can be of the same capacity or smaller or (preferably) larger as long it has adequate space to hold everything on the main card.
2. Shutdown the Pi and remove the main SD card.
3. Install this new SD card into the Pi in place of the original card, boot as normal, and install Raspian.
4. Shutdown the Pi and remove the duplicate SD card.
5. Reinstall the original SD card.
6. Start the Raspberry Pi and login.
7. List the disks by using

```
ls -l /dev/disk/by-id
```

8. Place the duplicate card in a USB adapter and insert into the Raspberry Pi.
9. Repeat step 7 and identify the USB drive by the new device that appears.
10. Create the *backup_rpi* script from Appendix O: Custom Scripts

```
sudo nano /usr/local/bin/backup_rpi
(copy and paste the contents of the script from the appendix.)
(Edit the clone variable and set it to the value found in step 9.)
(CTRL-X to save)
```

11. For automatic backup, setup a ***cron*** job to run the script periodically. I recommend every 2-3 months to not wear out the SD card. The normal daily, weekly, and monthly backup will capture in between changes.

```
sudo nano /etc/cron.d/sysop
```

12. Add these lines

```
# SD CARD (OS) backup the first of every quarter @ 11:35PM (before updates)...
35    23    1     1,4,7,10   *     root  /usr/local/bin/croncall backup_rpi
```

See the

Strings

Because the commandline parses by space between tokens, fields containing spaces must be quoted. Often, it does not matter whether using single quotes or doubles quotes, but as a rule, generally the shell (i.e. parser) cannot see inside single quotes but can see inside double quotes. This means variable substitutions inside strings generally require double quotes and to protect again misinterpreting content as special requires single quotes and/or escaping.

Comments

Commands prefixed with a ***#*** character represent comments for Bash shell operations. While not common for interactive use, comments commonly occur throughout scripts to document actions of the script.

> ***Tip: Other scripting languages may define a different specific comment character.***

Man Pages – Linux Help

The Linux manual exists online for access interactively while working from the commandline.

```
man cat
```

Typing ***man*** followed by a command as in this example will open the help pages for that command in the current shell terminal. The user can scroll or page up and down to move around. When finished type ***Q*** or ***q*** to quit and return to the shell. You can also access man pages online from any of a number of web sites.

Program Installation

apt-get

The apt-get command, short for aptitude get where aptitude represents the name of the code Linux code repository installer, installs or removes pre-packaged software builds. It also implements commands to update code references and upgrade already installed packages.

Cron Jobs section of *Appendix M: Linux Jumpstart* for details on how to alter these settings.

> ***Tip: Run this script manually prior to any major server changes such as distribution upgrades.***

> ***Tip: Running this script prior to updates ensures a fallback to the previous installation should anything go wrong with the update.***

(MS Windows) Tools for Data Backup and Management

Scanner

Scanner provides a graphical image of disk usage to locate large resource users quickly.

Robocopy

Essentially a Windows equivalent to the Linux ***rsync***, ***Robocopy*** provides a commandline copy tool with many options for doing complex copying locally or across a network. See robocopy help for usage.

Richcopy

A GUI frontend for ***robocopy***. Easier use than the commandline tool for novices, but does not support all ***robocopy*** options.

Dedicated Services

Everything described to this point represent basic network services applicable to any network and server regardless of use. While every server does not necessarily require everything, the mentioned services certainly represents a checklist of things to consider in setting up every server and at least one server on the local network must provide the services. This section describes other specific dedicated services that the network may provide, including:

- (Active Directory) Domain Controller: Handles user login access the network.
- File and Printer Sharing: Provide Windows compatible file and printer sharing services.
- Webserver: Host one or more websites.

Each of these may be optionally setup. Additionally, it may be desirable to use multiple servers for load balancing.

Samba

Samba provides emulation of the MS Windows SMB protocol, which supports a variety of services with the two of interest being Active Directory Services and file and printer sharing.

> ***Warning: The Samba Team does not presently recommend using an Active Directory server as the file server so the file server may require a second machine if you incur BIND problems.***

Follow the instructions in *Appendix Q: Building Samba* to build it from source, otherwise install from the repository. Do not do this step if you build samba locally.

> ***NOTE: As of*** Stretch***, the repository contains a more recent build of Samba and I do not recommend building it from source.***

```
sudo apt-get –y install samba
sudo samba –V
```

This should report the version of samba installed as version 4.x.x.

> ***Warning: A repository install places samba in the*** /bin ***folder with config files in*** /etc/samba ***folder. An install built from source places things in*** /usr/local/samba***.***

Skip to the *Active Directory Server* section.

For a local build, add the samba folders (i.e. /usr/local/samba/bin/ & /usr/local/samba/sbin/) to the PATH by editing ***/etc/login.defs***.

```
sudo nano /etc/login.defs
(Change PATH as follows…)
#ENV_SUPATH      PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin
ENV_SUPATH      PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin \
:/usr/local/samba/sbin/:/usr/local/samba/bin/
(CTRL-X to save)
```

NOTE: \ at end of line denotes line continuation.

You may need to also edit the ***/etc/profile*** file as below, comment out the existing PATH definitions and replace with those given.

```
sudo nano /etc/profile
#PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin: \
     /usr/local/samba/sbin:/usr/local/samba/bin"
#PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/local/games: \
     /usr/games"
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin: \
     /usr/local/samba/sbin:/usr/local/samba/bin"
(CTRL-X to save)
```

Logout and back in to launch a new shell for testing. Then run

```
sudo samba –V
```

This should report the version of samba installed as version 4.x.x.

Active Directory Server

Directory services function similar to a phonebook by providing network services a means to query about users, machines, devices, etc. ***Active Directory*** is Microsoft's directory service that works across Windows platforms. The Active Directory service functions as a single point of management of computers and user logins using Kerberos password management.

These directions follow essentially those provided by the Samba 4 Wiki documentation for setup of an Active Directory server with some additional notes and changes. The Active Directory setup uses the following site-specific definitions:

```
Realm:                          home.saranamabq.net
Domain:                         home
Server Role:                    dc
DNS backend:                    SAMBA_INTERNAL
DNS forwarder IP Address:       192.168.0.251
Administrator password:         ***********

AD domain name:                 home.saranamabq.net
AD DNS name:                    home.saranamabq.net
AD Kerberos realm:              HOME.SARANAMABQ.NET
NetBIOS domain name:            HOME
Domain Administrator:           HOME\Administrator
AD DC hostname:                 tiny, tiny.home.saranamabq.net
Samba Server IP address:        192.168.0.253
```

Once installing Samba, the first step to setting up an Active Directory server involves *provisioning* the server, a kind of initialization process that builds the ***smb.conf*** file. The samba-tool suite supports provisioning via an interactive session using the information provided above. Simply run as superuser…

```
su
samba-tool domain provision --use-rfc2307 --interactive
```

Copy the (final) output of the provisioning, specifically the SID, into a file named samba_id_info.txt for future reference. Perform the setup changes outlined in the *Active Directory DNS Configuration* section below. Then verify that everything works correctly by running the tests from the Samba Wiki:
https://wiki.samba.org/index.php/Setting_up_Samba_as_an_Active_Directory_Domain_Controller

Assuming the tests pass (indicating that Samba, Kerberos, and DNS work properly), see the

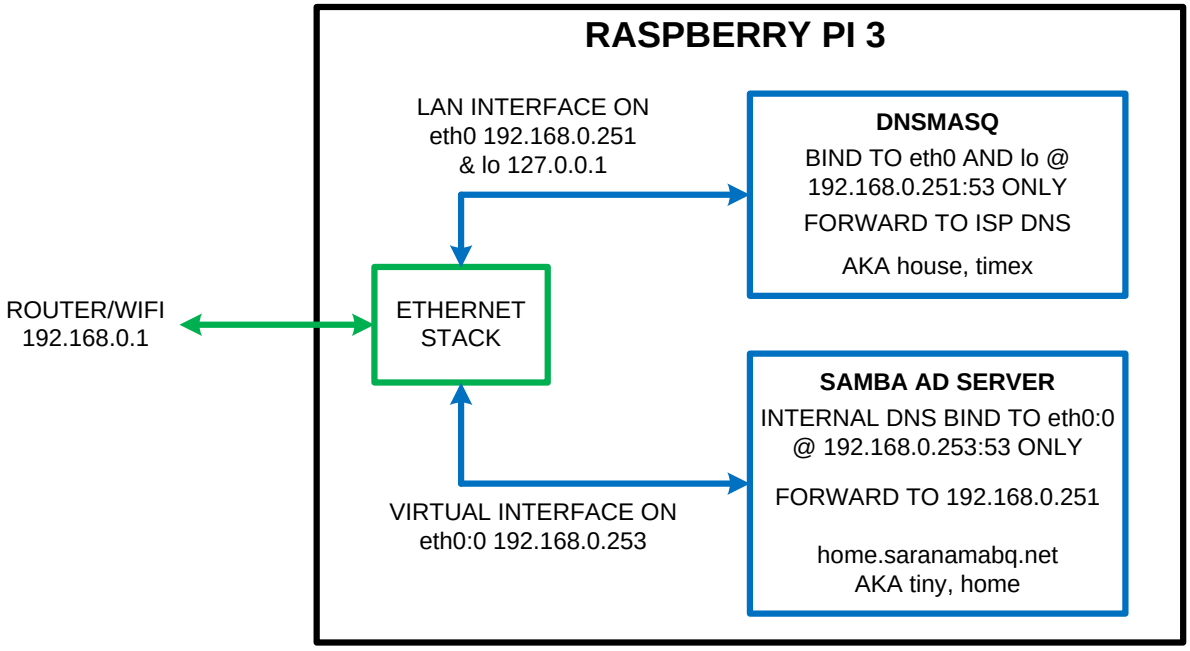

Figure 2: Active Directory DNS Configuration

Edit the Samba configuration

```
sudo nano /etc/samba/etc/smb.conf
```

Or for a local build

```
sudo nano /usr/local/samba/etc/smb.conf
```

Add these 2 lines to the [global] section:

```
interfaces = eth0:0
bind interfaces only = yes
(CTRL-X to save)
```

Then edit the DNSMasq configuration

```
sudo nano /etc/dnsmasq.d/saranam.conf
```

Add these 2 lines:

```
interface=eth0
bind-interfaces
(CTRL-X to save)
```

> ***Tip: Note the different interface specifications and keywords for these two configurations.***

Restart both services. Run the ***netstat*** command to verify that both DNS services have started.

```
sudo netstat -tapn | grep ":53"

tcp   0    0    192.168.0.253:53          0.0.0.0:*      LISTEN  20890/samba
tcp   0    0    127.0.0.1:53              0.0.0.0:*      LISTEN  18399/dnsmasq
tcp   0    0    192.168.0.251:53          0.0.0.0:*      LISTEN  18399/dnsmasq
tcp6  0    0    ::1:53                    :::*           LISTEN  18399/dnsmasq
```

```
tcp6  0     0     fe80::ba27:ebff:fe0b:53     :::*          LISTEN  18399/dnsmasq
```

Warning: Samba currently only understands IPv4. The gateway must be configured to NOT send IPv6 DNS info to DHCP clients or DNS will not work properly.

File and Printer Sharing for Windows

Samba supports the Microsoft Windows file and printer sharing service noted here for completeness.

Warning: Due to DNS issues, the Samba Team does not recommend running a Windows File and Printer Server on the same device as the Domain Controller. As an alternative, any Windows machine on the network, such as an admin management node, can easily provide the same services with domain level control.

Common UNIX Printing System (CUPS)

The common UNIX printing system established a singular way of handling network printing. Once installed, you can add printers easily to the CUPS server via a web interface at port 631. CUPS handles the print queues and access for each printer. After installing, add pi to the lpadmin (i.e. line printer administrators) group too.

```
sudo apt-get install cups
sudo usermod -a -G lpadmin pi
```

Then edit the daemon configuration to allow administration from other machines on the local network by adding a line to the listen section as given below:

```
sudo nano /etc/cups/cupsd.conf
# in the listen section add the line after the following line
# Only listen for connections from the local machines.
Listen 192.168.0.0:631
(CTRL-X to save)
```

Then enable remote administration and restart the server

```
sudo cupsctl --remote-admin
sudo /etc/init.d/cups restart
```

Then browse… http://localhost:631 ***or*** http://tiny:631

You can add a PDF printer to CUPS by

```
sudo apt-get install cups-pdf
```

Then edit ***/etc/cups/cups.conf*** and comment out the line that defines where output files get placed and replace with the appropriate line such as follows...

```
sudo nano /etc/cups/cups-pdf.conf
#Out ${HOME}/PDF
Out /mnt/datos/tmp/PDF
(CTRL-X to save)
```

Then using a browser on the local network go to ***http://192.168.0.253:631*** or ***http://tiny:631***. Click on ***Adding Printers and Classes*** and provide the user ***pi*** login credentials. Select local printer ***CUPS-PDF (Virtual PDF Printer)***. Click Continue. Edit the descriptions as desired. I recommend placing the output

path in the location. Select Share This Printer. Click Continue. Select a Make and Model. (It doesn't matter really, I use Lexmark, E260DN, same my home printer.) Click Add printer.

You can follow this procedure to add any other network printers and serve them across the network.

Avahi

The Avahi service emulates Apple Air Print. If installed on the CUPS server, its printers will be visible to Apple platforms. Install the Avahi service by

```
sudo apt-get install avahi-daemon
sudo update-rc.d avahi-daemon defaults
```

Microsoft Windows Management Tools section below for setup up the Active Directory.

Active Directory DNS Configuration

Active Directory requires a more sophisticated DNS service than supported by DNSMasq. Samba (internal-DNS) includes the necessary service but the setup requires some modification to the DNSMasq and Samba configurations.

The basic notion involves defining Samba Internal DNS as the primary service. It forwards unknown requests to DNSMasq, which in turn forwards requests to your ISP service. Figure 2 illustrates this arrangement.

In order to achieve the configuration we use a virtual network interface to provide a specific IP address for Samba separate from the host DNSMasq address. (See *DNSMasq* section for details.) Then we bind each specific DNS service on port 53 to only the specific interfaces assigned to the addresses. (Note: DNSMasq automatically binds to the local loopback address, i.e. 127.0.0.1, too!)

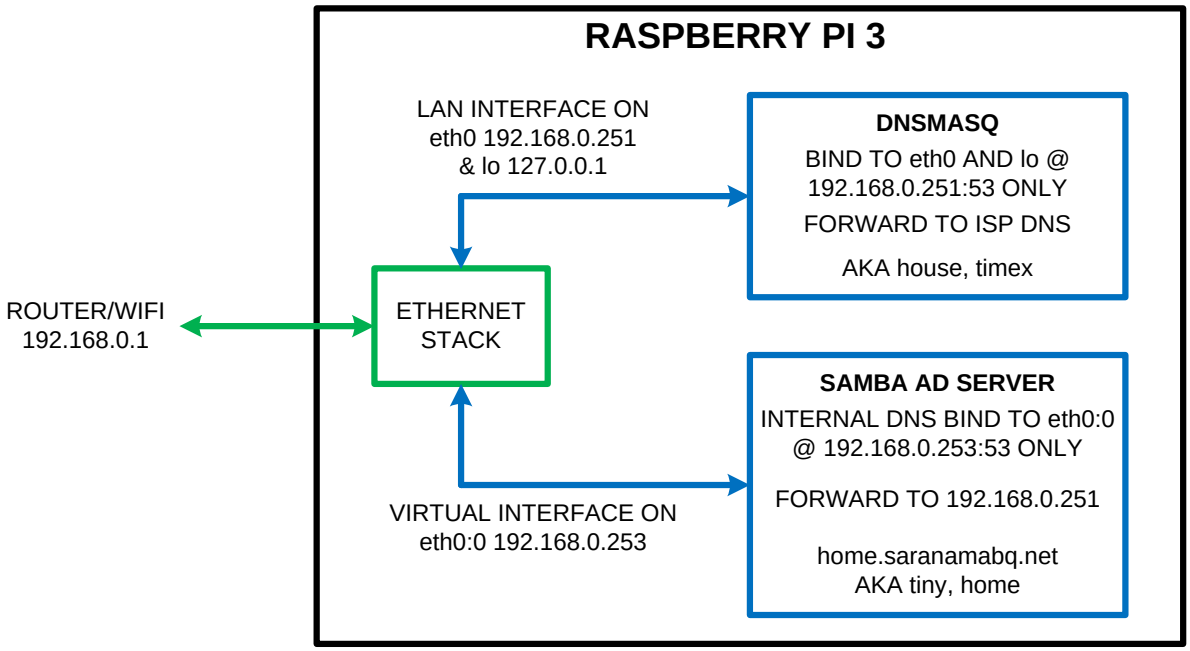

Figure 2: Active Directory DNS Configuration

Edit the Samba configuration

```
sudo nano /etc/samba/etc/smb.conf
```

Or for a local build

```
sudo nano /usr/local/samba/etc/smb.conf
```

Add these 2 lines to the [global] section:

```
interfaces = eth0:0
bind interfaces only = yes
(CTRL-X to save)
```

Then edit the DNSMasq configuration

```
sudo nano /etc/dnsmasq.d/saranam.conf
```

Add these 2 lines:

```
interface=eth0
bind-interfaces
(CTRL-X to save)
```

> ***Tip: Note the different interface specifications and keywords for these two configurations.***

Restart both services. Run the ***netstat*** command to verify that both DNS services have started.

```
sudo netstat -tapn | grep ":53"

tcp   0    0    192.168.0.253:53          0.0.0.0:*      LISTEN  20890/samba
tcp   0    0    127.0.0.1:53              0.0.0.0:*      LISTEN  18399/dnsmasq
tcp   0    0    192.168.0.251:53          0.0.0.0:*      LISTEN  18399/dnsmasq
tcp6  0    0    ::1:53                    :::*           LISTEN  18399/dnsmasq
tcp6  0    0    fe80::ba27:ebff:fe0b:53   :::*           LISTEN  18399/dnsmasq
```

> ***Warning: Samba currently only understands IPv4. The gateway must be configured to NOT send IPv6 DNS info to DHCP clients or DNS will not work properly.***

File and Printer Sharing for Windows

Samba supports the Microsoft Windows file and printer sharing service noted here for completeness.

> ***Warning: Due to DNS issues, the Samba Team does not recommend running a Windows File and Printer Server on the same device as the Domain Controller. As an alternative, any Windows machine on the network, such as an admin management node, can easily provide the same services with domain level control.***

Common UNIX Printing System (CUPS)

The common UNIX printing system established a singular way of handling network printing. Once installed, you can add printers easily to the CUPS server via a web interface at port 631. CUPS handles the print queues and access for each printer. After installing, add pi to the lpadmin (i.e. line printer administrators) group too.

```
sudo apt-get install cups

sudo usermod -a -G lpadmin pi
```

Then edit the daemon configuration to allow administration from other machines on the local network by adding a line to the listen section as given below:

```
sudo nano /etc/cups/cupsd.conf
# in the listen section add the line after the following line
# Only listen for connections from the local machines.
Listen 192.168.0.0:631
(CTRL-X to save)
```

Then enable remote administration and restart the server

```
sudo cupsctl --remote-admin

sudo /etc/init.d/cups restart
```

Then browse… http://localhost:631 ***or*** http://tiny:631

You can add a PDF printer to CUPS by

```
sudo apt-get install cups-pdf
```

Then edit ***/etc/cups/cups.conf*** and comment out the line that defines where output files get placed and replace with the appropriate line such as follows...

```
sudo nano /etc/cups/cups-pdf.conf
#Out ${HOME}/PDF
Out /mnt/datos/tmp/PDF
(CTRL-X to save)
```

Then using a browser on the local network go to ***http://192.168.0.253:631*** or ***http://tiny:631***. Click on ***Adding Printers and Classes*** and provide the user ***pi*** login credentials. Select local printer ***CUPS-PDF (Virtual PDF Printer)***. Click Continue. Edit the descriptions as desired. I recommend placing the output path in the location. Select Share This Printer. Click Continue. Select a Make and Model. (It doesn't matter really, I use Lexmark, E260DN, same my home printer.) Click Add printer.

You can follow this procedure to add any other network printers and serve them across the network.

Avahi

The Avahi service emulates Apple Air Print. If installed on the CUPS server, its printers will be visible to Apple platforms. Install the Avahi service by

```
sudo apt-get install avahi-daemon
sudo update-rc.d avahi-daemon defaults
```

Microsoft Windows Management Tools

> ***Info: This section actually does not involve the Raspberry Pi server directly, but identifies tools run on a Microsoft Windows machine for managing the server, remotely. While not required, use of the tools greatly simplifies management of users and policies. The tools may be installed on any MS Windows based machine on the local network, although a dedicated administrator machine may be safer and more secure.***

Once you setup and test the Active Directory domain controller, you can manage it remotely using ***Remote Server Administration Tools*** (RSAT) supported across Windows platforms. These tools allow administrators to create user accounts, reset passwords, join machines to the domain and any other services setup normally needed.

Download

To acquire the tools go to http://windows.microsoft.com/en-us/windows/downloads and search for ***Remote Server Administration Tools***. Select the version of Windows that you wish download.

Remote Server Management Setup

1. Install the RSAT tools on a Windows machine.
2. Attach the Windows machine to the same network as the Active Directory server.
3. Log in to the Windows machine as a local administrator.
4. Join the machine to the domain. (Instructions vary specific to Windows version.)

   For Windows 10, press the **Windows Key** and type **This PC**. Right-click on **This PC** from the list and select **Properties**. Then **Advanced system settings**. Then click on the **Computer Name** tab in the dialog and click **Change**. Click **Domain** and type ***Home***. Then provide the **Administrator** credentials defined when setting up Samba.

5. Reboot.
6. Log into the Windows machine using the domain\Administrator credentials.
   i.e. ***HOME\Administrator***

The domain is ready for use and join other network machines to the domain.

Active Directory Users and Computers

The Active Directory Users and Computers application of RSAT provides the means to add new users, groups, and computers to the domain. See *Appendix T: Active Directory Users and Computers* for more instructions for completing a specific user setup.

1. Start the ***Active Directory Users and Computers*** application.
   a. Press the **Windows Key** and type ***Active Directory …***. Click on it from the list.
   b. You can right-click and select **Pin to Taskbar** to make future invocations easier.
2. Select the domain in the left pane, i.e. ***home.saranamabq.net***.
3. From the menu, select **Action** | **New** | **User** or click on the **New User Icon** from the toolbar and begin adding user accounts.
4. Likewise, select Action | New | Group and define desired groups.
5. Add users to the required groups.
6. You can add computers most easily by joining the specific machine to the domain.

Group Policy Management Console and Editor

Use of the **Group Policy Management Console (**GPMC) and **Group Policy Management Editor** (GPME) exceeds the scope of this document, but a brief discussion follows as an introduction.

> ***Warning: Policies and registry settings can render the domain nonfunctional. Backup your configuration first! I recommend you do not tamper with preconfigured or default policies.***

The GPMC of RSAT allows administrators to define and manage policies for the domain computers and users. You can write blanket policies or apply them to specific computers and users, which makes management more flexible.

Organizational Units

The Saranam domain implements organizational units (OU) within Active Directory for group policy management including:

1. ***Saranam Instructors and Clients***: Only Saranam instructors, guest speakers, and students.
    a. ***Test***: Special sub-OU for admin testing.
2. ***Saranam Staff***: Only Saranam staff.
    a. ***Test***: Special sub-OU for admin testing.
3. ***Saranam Admins***: Domain Admins.

Group Policy Objects

Group policy definitions exceed document scope, but for reference, the Saranam domain manages the group policy objects across various OUs as outlined in the following table. (Subject to change.)

GPO	OUs	Description
Flash Drives Policy	Clients and Instructors	- Prohibits the use of removable media.
Legal Notice	Domain	- Displays a legal use notice before login.
One Drive	Domain	- Disables Microsoft One Drive for Storage.
Password Policy	Domain	- Defines use of strict passwords. - Sets password requirements. - Sets login failure timeouts.
RDP Allow Specific User	*"Test" users*	- *Enable RDP for special client and staff "test" users.*
Saranam Wallpaper	Admins, Clients and Instructors	- Sets the Desktop wallpaper.
Screensaver Policy	Domain	- Forces screensaver use, password, and timeout.
Time Service	Domain	- Defines network time server.
Windows Update, Reporting, etc	Domian	- Sets a number of Windows defaults for updates, error reporting, and such.

Group Policy Utilities

The shell commandline command (i.e. ***cmd.exe***) ***gpupdate*** applies policy changes immediately instead of waiting for normal propagation (i.e. reboot or logout or timeout). The ***gpresult*** command reports the cumulative policy for specific users and computers.

> ***Tip: Keep a backup and printed (PDF) copy of policies for recovery if they become corrupted.***

Web Services

The Raspberry Pi easily provides resources necessary for a small/medium webserver for both intranet and Internet use. The spectrum of possibilities itself goes well beyond the scope of configuring a small office server. Nonetheless, the following provides an example of setup to provide basic web services.

> ***WARNING: Web server operation involves significant security issues in protection of information and resources such as the server itself. I recommend a separate server machine to minimize compromise of the network operations server. Further, you may not want to run a webserver on the same box as your basic network/domain services as a webserver overload condition may directly impact network all operations.***

Servers

Apache

Apache represents the workhorse of web servers running some 80%+ of the Internet. While some have demonstrated running Apache on the Raspberry Pi, Apache uses a significant amount of resources and will not scale well, likely overwhelming the server in short order. Also setting up Apache can be challenging even for the experienced admin. I do not recommend running Apache on the Raspberry Pi.

Lighttpd

Lighttpd (pronounced light-dee) implements a lightweight web server with basic capabilities including support of Fast-CGI and PHP scripting. It's open source; it's free. This probably represents the simplest out of box setup for web services.

NodeJS

NodeJS provides server side JavaScript for building your own custom web service. See separate document entitled *A Flexible User Configurable NodeJS Web Server* for details.

Database Support

SQLite

SQLite implements a simple but very scalable database recommended for use with the NodeJS setup. To install run

```
sudo apt-get install sqlite3
```

Then create a pointer by

```
sudo ln -s /usr/bin/sqlite3 /usr/local/bin/sqlite
```

You can open any database by the command

```
sudo sqlite <database_name>
```

I recommend using the file extension ***.sq3*** to denote an SQLite 3 database and ***.sql*** for SQL syntax files.

Appendix M: Linux Jumpstart

What is Linux?

Linux is an open source operating system (OS) for a computer. The operating system performs basic functions that enable user interaction with the computer as well as interaction with other machines. Raspian represents a version of Linux specifically optimized for the limited resources and hardware of Raspberry Pi. It derives from the Debian kernel, which also provides the kernel of the popular Ubuntu, often referenced in Raspberry Pi tutorials.

The Linux Shell

Many users confuse the Linux shell with Linux. The shell is simply the program interface to the user that usually supports interaction with the keyboard and/or mouse and video display. Shells come in two forms: textual and graphical. Textual shells utilize a commandline for typing in commands as text strings. Non-Linux users familiar with the once popular DOS shell have already experienced a textual context shell. Linux can provide multiple shell types simultaneously in separate terminal windows. By default the Raspian flavor of Linux uses the *Bourne Again Shell*, commonly called the ***BASH*** shell. Graphical User Interfaces (GUI) such as Microsoft Windows or MIT's X-Windows are nothing more than fancy shells that understand how to interact with a mouse and provide a more visual experience to the user often making it easier for the user to interact. Because network server management generally deal with executing low-level operations, instructions generally involve entering commands in textual shells.

Some Linux Basics

Users, Superuser, Root, Groups, and World

We refer to the person actively interacting with the machine as a ***user*** who is a member of one or more ***groups***. When a user has successfully logged into the system a text shell will present them with a commandline prompt such as

```
pi@raspberrypi ~ $
```

This prompt tells the user the system is ready to take instructions to perform a given task for user *pi*. The ~ character represents shorthand for the user's home folder, in this case */home/pi*. By default, a user may only perform operations within their own user scope determined by system settings of group membership and ownership.

The user may or may not have special administrative privileges to perform special system wide controlling operations. We refer to a user having these special privileges as a ***superuser***. One specific user called ***root*** always has these special privileges. The ***superuser*** prompt follows a slightly different format using a hashtag instead of a dollar to distinguish its privilege as a system administrator as in

```
root@black:/home/pi#
```

Attempting to perform a superuser command without superuser privileges will give the familiar permission denied response as in

```
pi@black ~ $ touch /etc/test
touch: cannot touch `/etc/test': Permission denied
pi@black ~ $
```

Users may temporarily act as a ***superuser*** (if allowed by configuration as a member of the ***sudo*** group) by prefixing each command with ***sudo***, which stands for "*superuser do*" as in

```
pi@raspberrypi$ sudo cp /etc/fstab /etc/fstab.save
pi@raspberrypi$
```

Following the command execution, the same user prompt occurs. This allows for safe use of ***superuser*** powers only when needed to prevent accidental action. In some cases one may need to perform multiple successive commands and this practice becomes annoying. In such a case, the user may execute the ***su*** login command to turn on superpowers continuously as in

```
pi@raspberrypi$ su
Password:
root@black:/home/pi#
…
root@black:/home/pi# exit
pi@raspberrypi$ su
```

The password response prompts the user to enter root's password. If successful the shell changes to user root. Once complete, typing ***exit*** will restore operation to the previous user.

Alternately, a user can run the user's default shell as root using...

```
pi@raspberrypi$ sudo -s
```

> ***Tip: Any operation involving installation of software or system configuration will require the use of superuser privileges even if not noted by the instructions.***

As noted, each user may be a member of one or more ***groups***. Members of ***groups*** share permissions and authority without having to share credentials.

The term ***world*** refers to a *pseudo-group* of all users and groups outside the scope of ownership, but realistically limited to system authenticated users.

Ownership and Permissions

Every file or folder has an ***owner*** and ***group owner*** and specific ***read***, ***write***, and ***execute*** (i.e. rwx) permissions assigned to each as well as the ***world*** users. This means you can define read, write, and execute operations for any file or directory to the owner or a group or set it open to anyone in the world. For example, a home folder listing (i.e. `ls –l /home/pi/bin`) gives

```
-rwxr-xr-- 1 pi staff 0 Mar 6 11:30 /home/pi/bin/test
```

This shows one file (first character, d for directory) named test with owner rwx, group r-x, and world r-- privileges, commonly referenced by an octal representation, 754. It also shows owner pi and group owner staff. See man page help for ***chown*** and ***chmod*** commands for altering ownership and permissions, respectively.

> ***Tip: Mastering ownership and permissions represents a fundamental element of creating and maintaining a secure system!***

Everything is a File

First thing to remember in Linux is that everything is a file, or at least behaves as such. This may sound strange to those unfamiliar, but keeping this perspective in mind will help you learn Linux faster and make you more efficient with commands and resolving issues. With this perspective…

- Files store information, which may include program instructions.
- Files may be readable or not.
- Files may be writable or not.
- Files may be executable or not, as either compiled code or scripts that provide instructions for processing other files.
- Files process information in two modes: by blocks or character-by-character.
- Files support streaming meaning continuous input or output (i.e. reading and writing).
- Files may be containers of information, including containers of other files (i.e. directories or folders).
- Files may be pointers to actual containers, known as links.
- Files may be physical devices that behave as files, sometimes called special files or devices. For example, one can think of a printer as a write-only file.
- Files may be processes. Processes act like files by creating or consuming information, usually in real time, usually by streaming.
- Files may be sockets, special stream processes, which connect to other processes or even other machines for passing information.
- Files have a name, associated owner, group owner, and permissions that define who can use it.

But to the user, these items all look, and act like files!

File Structure

Linux uses a prescribed file directory or folder structure, actually several. Linux evolved from an amalgamation of several UNIX operating systems, including AT&T System 5, Berkeley Software Distribution (BSD), and even Linux original notions. As a result, the file system has elements sometimes overlapping all of them. **Error! Not a valid bookmark self-reference.** lists the normal Linux file system installed by Raspian. Users can only access files and folders owned by root with superuser permissions.

Folders denoted with an * represent pseudo or virtual folders such as external or temporary file system mount points that have special purpose and behavior. These folders do not need backed up for system restore and may in fact cause problems if backed up.

Links

Links, similar to MS Windows *shortcuts*, point to files. Links may be hard links, which represent multiple path references to the same physical file within the file system, but more typically symbolic, which simply add a pointer to the primary file reference. Use the ***ln*** command to create links. Listing a folder (***ls***) with the –l option will show the source of defined links. Hard and symbolic links behave differently for copy, move, and backup operations.

Table 1: Linux Filesystem

Folder		Owner	Use
/bin		root	System binaries storage, i.e. programs available to all users.
/boot		root	Holds boot files. (On RPi this mounts to the boot partition.)
/dev	*	root	Pseudo folder that maps hardware devices
/etc		root	System configuration data. Many subfolders specific to various programs such as /etc/samba
/home		<users>	Stores each user's profile, except root. (Equivalent to ~/ shortcut.)
/lib		root	Code libraries for compiling
/lost+found		root	Storage for disk error recovery fragments.
/media	*	root	Removeable media, i.e. floppy and USB, (auto) mount point.
/mnt	*	root	Network drive mount point.
/opt		root	Legacy system folder that holds optional packages.
/proc	*	root	Pseudo folder for processes
/root		root	Home folder for root.
/run	*	root	Pseudo folder for running programs state data.
/sbin		root	System binaries, storage, i.e. programs, with root access only.
/selinux	*	root	"Secure Linux" hold over. (empty)
/srv	*	root	System services data folder. (empty)
/tmp	*	-	Temporary storage open to all users. (Usually a mount to a temporary file system (i.e. tmpfs) such as a ram disk. It is erased or cleared on reboot.)
/usr		root	Machine specific program storage. /usr/local subfolder does not get overridden by OS updates ans so is generally used to store user customized scripts.
/var	*	root	Program variables data storage, such as logs, cache files, file locks, run states, and process info.

The Console

The keyboard and video screen or terminal window make up the *computer console* that provides the mechanism for user interaction. The keyboard refers to a special ***read-only*** file. The video screen or terminal window of the console represents a ***write-only*** file (although screen capture may provide a pseudo-read capability). The ***commandline*** refers to the input line of the terminal window where the user types commands to execute.

Newline Character(s)

The newline character refers to the character(s) terminating a line of text in a file, thus delineating the start of a newline. Pressing the ***Enter*** or (Carriage) ***Return*** key sends the newline character(s). The new line character differs across operating systems. Linux uses \n (character 0x0A); DOS and MS Windows uses \r\n (characters 0x0D 0x0A); Apple systems use \r (character 0x0D). While usually handled automatically and generally transparent to most operations across platforms, be aware it will matter in some cases.

> ***Tip: Linux scripts expect Linux newlines only. They will fail otherwise, so be careful with MS Windows to Linux copy-paste operations or MS Windows editing tools in particular.***

Escape Character

Some characters in a command may have special behavior such as regular expression patterns or strings. In such cases, it may be necessary to "escape" the characters by prefixing with the **** character to turn off their special behavior.

For example, the escape character in the single quoted string 'john\'s home' means ignore the string ending meaning of the second ' character to treat it as part of the value, as in "john's home".

Alternately, the **** character, as used above in the newline description, may designate the following character as having special meaning. That is, it escapes the character's normal default meaning. The notation \n indicates a special newline character, not the normal letter *n* character.

Basic Commands and Syntax

Commands and Scripts

Commands refer to input entered at the commandline of a terminal or console to perform various actions, such as copying a file. These can be compiled programs or interpreted scripts. Compiled programs generally run faster and keep their contents proprietary while scripts generally exist as simple text files written in some specific scripting language such as Perl, Python, Bash shell, etc. Unlike MS Windows where the file extension specifies the type of file and associated program for execution, the first line of a script, somethings called the hash bang because of the first two characters, references the interpreter used to run the script as in the following:

```
#!/bin/bash
```

Table 3 lists some commonly used BASH commands, their common options or switches, and their arguments. See each command's ***man*** page for complete information.

Syntax example…

```
command –switch <arg1> [<arg2>]
```

The line above illustrates syntax used throughout this document and commonly applied across tutorials. Each space delimited string is referred to a token or field. The command itself occurs first on the line. The order of the remaining fields depends on the specific command but usually doesn't matter, except for multiple arguments. The –switch term represents a command switch or option, usually prefixed by a – character. Switches alter the commands default behavior. Generally switches immediately follow the command. Most commands allow aggregated switches, meaning "-l -a" is the same as "-la" or even "-al". Arg1 and arg2 represent the command arguments such as filenames. The **<>** characters around these terms mean the name stands for the actual content rather than the literal string. For example, the argument <path> would expect an actual file path, not the word path. The ***[]*** characters identify optional arguments. For the example given, <arg1> would be required, but <arg2> is optional.

Compound Commands

You can concatenate Commands in various ways to simplify typing and extend operations from one command to the next. Table 2 gives a summary of common command operations. Note that virtually any combination of thee sequences may be used to perform very complex operations such as controlling a script with control input from a file, piping its output to a second script such as a filter whose output is redirected to a second file.

Table 2: Linux Command Operations

Character	Operator	Function	Example
&	Background	Executes current command in the background and returns to command prompt immediately.	`run_a_test &`
|	Pipe	Takes output of one command into the input of the next.	`ls | grep "pi"`
<	Input Redirect	Redirects a file into a commands input.	`grep "pi" < /tmp/list`
>	Output Redirect	Redirects a command output into a file.	`ls > /tmp/list`
>>	Append Redirect	Redirects a command output to append to a file.	`ls >> /tmp/list`
2>&1	Redirect All	Redirect a commands error to output stream.	
2>&1|	Pipe All	Pipe a command's error and output.	
;	Command separator	Concatenates a series of command in the same line	`cd /tmp; ls`
()	Group	Treat commands as a group with respect to input and output.	`(pwd; ls –l) > out`

Strings

Because the commandline parses by space between tokens, fields containing spaces must be quoted. Often, it does not matter whether using single quotes or doubles quotes, but as a rule, generally the shell (i.e. parser) cannot see inside single quotes but can see inside double quotes. This means variable

substitutions inside strings generally require double quotes and to protect again misinterpreting content as special requires single quotes and/or escaping.

Comments

Commands prefixed with a ***#*** character represent comments for Bash shell operations. While not common for interactive use, comments commonly occur throughout scripts to document actions of the script.

> ***Tip: Other scripting languages may define a different specific comment character.***

Man Pages – Linux Help

The Linux manual exists online for access interactively while working from the commandline.

```
man cat
```

Typing ***man*** followed by a command as in this example will open the help pages for that command in the current shell terminal. The user can scroll or page up and down to move around. When finished type ***Q*** or ***q*** to quit and return to the shell. You can also access man pages online from any of a number of web sites.

Program Installation

apt-get

The apt-get command, short for aptitude get where aptitude represents the name of the code Linux code repository installer, installs or removes pre-packaged software builds. It also implements commands to update code references and upgrade already installed packages.

Cron Jobs

The Linux cron command provides an extremely flexible and powerful way to run tasks on schedule, such as periodic backups and OS updates. All files placed in the ***/etc/cron.d*** folder will be treated as cron jobs by the OS. Each task has parameters for the time to run, the owner, the command to execute, and any parameters for the command.

Run time parameters use whitespace delimited fields for ***min***, ***hour***, ***day_of_month***, ***month***, and ***day_of_week*** (where Sunday=0 or 7). A * wildcard may be used to match any time. For example,

```
# OS update @ 9:35PM every Friday...
35      21      *       *       5       root    /usr/local/bin/croncall update
```

(Here /usr/local/bin/croncall just represents a wrapper that logs cron calls and is not required.)

Time fields may also use a comma delimited list of parameters. For example, the day_of_month could be specified as 1,15 (note, no whitespace between numbers) to run on the 1ST and 15TH of the month.

Some special predefined terms specify certain run conditions such as @reboot to run a job every time the machine starts. See ***man cron*** for more information.

Client/Server Model

The client server model defines a common communications model between two machines. The client machine represents the machine making or initiating requests for information and the server represents the machine fulfilling the request. For example, a machine running a web browser represents a client. The machine responding to that request represents the web server.

Daemons

In Linux ***Daemons*** are processes that usually run in the background to provide services needed for operation of the server, clients, and network. Examples include DNS, DHCP server, NTP, … These daemons often run with representative names suffixed with the letter d, for example, ***ntpd*** provides the NTP daemon service.

A special process starts and stops a daemon so that it continues to run after the interactive session that started the daemon stops. Most any process may be turned into a daemon, which allows users to create custom services. Raspian typically uses a SYS5 (i.e. AT&T UNIX) style ***init*** script to start, stop, restart, or check the status of a daemon as in …

```
sudo /etc/init.d/daemonx start
sudo /etc/init.d/daemonx status
sudo /etc/init.d/daemonx restart
sudo /etc/init.d/daemonx stop
```

Alternately, the process may use the Ubuntu style service command as in…

```
sudo service daemonx start
```

Best Practices

Backup and Updates

Obviously, keeping data backed up and the OS up to date stand as two of the most important system administrator tasks. Automating such important tasks makes sure they occur on a timely basis. See the *Automatic Server Update* and *Automatic* Server Update

0

See also Appendix C: Custom Scripts for useful “embellished” commands, such as pid, strip, update.

Appendix N: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER.

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

Custom Scripts includes an update script for automatically keeping the server up to date.

Create the update script by

```
sudo nano /usr/local/bin/update

(Copy the contents of the /usr/local/bin/update script from 0

See also Appendix C: Custom Scripts for useful "embellished" commands, such as pid,
strip, update.
```

Appendix O: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER***.***

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

```
Custom Scripts into the file and save.)
```

> ***Tip: Save your scripts in /usr/local/bin. The operating system preserves this area on upgrades.***

Then establish the ***cron*** tasks by

```
sudo nano /etc/cron.d/sysop
```

Add these lines

```
# OS update @ 9:35PM every Friday...
35      21      *       *       5       root    /usr/local/bin/croncall update
```

See the

Strings

Because the commandline parses by space between tokens, fields containing spaces must be quoted. Often, it does not matter whether using single quotes or doubles quotes, but as a rule, generally the shell (i.e. parser) cannot see inside single quotes but can see inside double quotes. This means variable substitutions inside strings generally require double quotes and to protect again misinterpreting content as special requires single quotes and/or escaping.

Comments

Commands prefixed with a **#** character represent comments for Bash shell operations. While not common for interactive use, comments commonly occur throughout scripts to document actions of the script.

> ***Tip: Other scripting languages may define a different specific comment character.***

Man Pages – Linux Help

The Linux manual exists online for access interactively while working from the commandline.

```
man cat
```

Typing ***man*** followed by a command as in this example will open the help pages for that command in the current shell terminal. The user can scroll or page up and down to move around. When finished type ***Q*** or ***q*** to quit and return to the shell. You can also access man pages online from any of a number of web sites.

Program Installation

apt-get

The apt-get command, short for aptitude get where aptitude represents the name of the code Linux code repository installer, installs or removes pre-packaged software builds. It also implements commands to update code references and upgrade already installed packages.

Cron Jobs section of *Appendix I: Linux Jumpstart* for details on how to alter these settings.

> ***Tip:*** cron ***processes all files it finds in the /etc/cron.d folder, so you can name it anything. Here sysop denotes system operations.***

With this setup, the system will automatically update every Friday at 9:35 PM and send the system administrator an email confirming the operation. The notification email displays a message indicating if the server requires a reboot, captures errors, and includes a log of the updates.

Backup sections for how-to details.

Configuration Changes

Before making any configuration changes always backup what you plan to change. For files a copy can be made of the original file with an altered name such as

```
pi@raspberrypi$ sudo cp /etc/network/interfaces /etc/network/interfaces.orig
```

This copies the original file into a backup with a .orig suffix. It the event of a problem you can restore the original settings by reversing the operation and copying the backup file to the configuration file.

Most configuration files recognize the # or other character as a comment. So when editing configurations a good practice involves commenting out an existing line and replacing with a new line such that record of the previous configuration remains in the file.

When multiple individuals work on the same configuration, it is useful to date and comment by user the changes made.

Dates

I recommend using YYYY-MM-DD [HR-MN-SS] format for dates, which sort chronologically and removes month/day ambiguity. For example, appending a date string such as ".20150814" to files will result in an ordered directory listing making it easier to sort out the latest version. It also provides an ordered historical record for recovery.

Warning: Do not use colons in date strings in filenames since it will not port to MS Windows.

Change Log

Administrators should keep a change log for servers. This helps links symptoms and causes when someone notes problems. It is particularly helpful for situations involving multiple administrators as a way of communicating who is doing what.

References

[1] B. Ward, How Linux Works, No Starch Press, ISBN 1-59327-035-6, 2004.
Provides a very good overview of many Linux operations and internal workings.

[2] J. Kemp, Linux System Administration Recipes, Apress, ISBN 978-1-4302-2448-5, 2009.
Provides solutions to many common administrative tasks.

Table 3: Some Common (BASH) Linux Commands

Command	Switches	Parameters	Description
alias		aka=command	Create a command alias
apt-get			Tool to install or remove software.
cat		list of files	Concatenate (stream, print) files to standard output.
cd		folder	Change directory to specified folder.
chown	-R	owner:group path	Change owner and group for file or folders (recursively)
chmod	-R	permissions	Change file permissions for file or folder (recursively) Permissions owner-group-world, read=4,write=2,execute=1
cp	-R	source dest	Copy a file from the source location to destination.
df		disk	Display the amount of disk space used.
echo		prompt	Print prompt to standard output.
exit			Terminate the current shell.
ln	-s	source link	Creates a symbolic link for specified source.
ls	-alR	[<path>]	List contents of a folder, optionally specified by <path>.
man		command	Help manual for the specified command.
mkdir		folder	Create a folder.
more			Paginate standard input to standard output (pipe to more)
mv		source dest	Move a file from the source location to destination.
nano (vi)		file	Text editor
nslookup (dig)		node	Lookup the IP address for a specified machine (node).
passwd		user	Change the passwd for the specified (or current) user.
ps	-aux		List process IDs.
reboot		now	Reboot the machine.
rm	-R	source	Remove a file
rmdir		Source folder	Remove a folder. (Must be empty.)
rsync			Remote sync or backup command.
shutdown			Halt the machine.
ssh		user@node	Secure shell remote login to user at a specific machine.
su			Start a superuser shell
sudo			Run next command as superuser
tar (bzip)			Archive command.
uname			Display the name of the current OS.

See also Appendix C: Custom Scripts for useful “embellished” commands, such as pid, strip, update.

Appendix P: Linux User Tips

.bashrc

The ***.bashrc*** file in each user's home folder (i.e. ***/home/<user>*** or ***~/***) can be edited to add or change personal preferences to the shell experience that become permanent each time the user logs in.

> ***Tip: In Linux, files prefixed with a*** . ***character become hidden files. Use*** ls -a ***to see them.***

> ***Tip: In Linux ~/ defines a shortcut to a user's home folder, same as /home/$USER.***

Alias

At the end of the ***.bashrc*** file are a list of predefined aliases using the bash alias command. Add the following to make shortcuts for a long listing and to list all files.

```
alias ll='ls -l --color=auto'
alias la='ls -a --color=auto'
```

/etc/profile

The ***/etc/profile*** file defines system wide bash shell setup. Changes to it will apply to all users. Changes to the /etc/profile require a reboot to take effect.

/etc/login.defs

The ***/etc/login.defs*** file defines login defaults. Use it to make changes to the PATH for users or superuser. Changes to the /etc/login.defs require a reboot or re-login to take effect.

Environment Variables

Environment variables get inherited by users and processes to pass configuration information as needed. Environment variables get defined in a number of locations by the precedence of ***/etc/login.defs***, ***/etc/environment***, ***/etc/profile***, and ***~/.bashrc*** with each later location overriding the previous. The first locations apply to all users while the .bashrc applies only to the specific user in whose home directory it is located.

PATH

The ***PATH*** variable defines the search order and locations that the shell uses to locate commands. The ***/etc/login.defs*** file initializes the path for both users and superuser. Override in ***/etc/environment*** and ***/etc/profile*** for all users, or in ***~/.bashrc*** for user specific changes.

You can (conditionally) add a local user ***~/bin*** to the path by including this at the end of the ***.bashrc*** file.

```
# add local user path if exists...
if [ -e ~/bin ]; then
    PATH=$PATH:~/bin
fi
```

type

Sometimes in may be confusing as to where the system locates a particular executable. The ***type*** command provides a means of querying a bash shell to trace the path for locating any command, for example

```
type tmux
```

reports

```
tmux is /usr/local/bin/tmux
```

This indicates that the current user retrieves the ***tmux*** executable from the ***/usr/local/bin*** folder.

TAB TAB

When you cannot remember the exact name of a command starting the command name and typing the TAB key twice will recall all available commands matching the pattern. Typing

```
raspi <TAB> <TAB>
```

For example, will return

```
raspi-config  raspistill    raspivid      raspiyuv
```

Suggested "Tiny" Links

You will find that system operations often involve using a few particular folders repeatedly. As such, it becomes useful to create simple links as typing shortcuts, such as listed in Table 4.

Then for example, any time you want to edit a script, instead of typing

```
sudo nano /usr/local/bin/my_script
```

You can instead type

```
sudo nano /i/my_script
```

Since Linux working usually involves heavy commandline interaction, minimizing typing has long been a hallmark of Linux best practice.

> ***Tip: Similarly, ~/ maps to*** /home/$USER***.***

Table 4: Suggested Shortcuts

Link	Folder	Command	Example Use
/u	/usr/local/bin	ln -s /usr/local/bin /u	sudo nano /u/update
/i	/etc/init.d	ln -s /etc/init.d	/i/dnsmasq start

Appendix Q: Custom Scripts

Tip: When copying scripts be sure to paste as simple test, not Windows Rich Text Format. Characters such as quotes and hypens get mangled.

NOTE: Be sure to use Linux newlines only (i.e. character 0x0A) for line endings, not DOS/Windows carriage return, linefeed characters (0x0D, 0x0A), or Mac newline character (0x0D) as this will result in a confusing runtime error "bad interpreter: No such file or directory".

NOTE: Normally customs scripts get placed in /usr/local/bin (with data in /usr/local/etc). The system preserves files in this area during updates, which ensures they do not get overwritten.

NOTE: When creating scripts in Linux you must set the file permissions using chmod, typically 755 for root edit and all user execute, and you may have to change the owner to root:users too using chown. Otherwise, the scripts will not execute even if defined correctly.

/usr/local/bin/backup

Script to run backup "lists" and notify admin. Lists consist of a series of rsync commands specifying files and folders backed up from a source area to a destination area.

```
#!/bin/bash

# linux backup script called from croncall...

# script variables
# exported variables and environment variables can be substited in list files
e=`date +%s`
logRoot=/tmp/backup.log
log="$logRoot.$e"
list=${1:-daily}
export host=`cat /etc/hostname`
export mnt=${2:-/mnt/usb/backup}
export backup="$mnt/$host"
export day=`date +%A`
export mday=`date +%d`
export week=$((10#`date +%W` % 4))
export month=`date +%B`
export year=`date +%Y`

if [ "$list" = "daily" ]; then
  flist=/usr/local/bin/backup.lst.daily
  fi
if [ "$list" = "weekly" ]; then
  flist=/usr/local/bin/backup.lst.weekly
  fi
if [ "$list" = "monthly" ]; then
  flist=/usr/local/bin/backup.lst.monthly
  fi
if [ "$list" = "test" ]; then
  flist=/usr/local/bin/backup.lst.test
  fi
#echo "LIST: $list"

echo "### [$USER] BACKUP BEGUN: `date`" >> $log 2>&1
echo "# HOST:  $host" >> $log 2>&1
```

```
echo "# LIST:  $list ($flist)" >> $log 2>&1
echo "# DATE:  Week $week - $day, $mday $month $year" >> $log 2>&1
echo  >> $log 2>&1


# handle mounting of backup drive...
mounted=0
try=6
n=0

if [ $mnt = "offline" ]; then
  disk="/dev/disk/by-label/offline"
else
  # assume using an existing locally mounted drive.
  mounted=-1
  fi

echo "# DRIVE: $disk" >> $log 2>&1
echo "# MOUNT: $mnt" >> $log 2>&1

if [ $mounted -eq 0 ]; then
  while [ $n -lt $try ];
    do
      #mount backup drive
      mount -t ext4 $disk $mnt >> $log 2>&1
      sleep 1
      if ! grep -qs "$mnt" /etc/mtab
      then
        # failed to mount, retry
        (( n = $n + 1 ))
        #echo "###   BACKUP DRIVE[$disk] failed to mount on attempt $n."
        echo "###   BACKUP DRIVE[$disk] failed to mount on attempt $n." >> $log 2>&1
        sleep 9
      else
        #echo "###   BACKUP DRIVE[$disk] mounted."
        echo "###   BACKUP DRIVE[$disk] mounted." >> $log 2>&1
        sleep 1
        ls -l $mnt >> $log 2>&1
        (( n=$try ))
        mounted=1
        fi
    done
  fi

if [ $mounted -eq 0 ]; then
  echo "### BACKUP DRIVE[$disk] NOT READY: BACKUP ABORTED!" >> $log 2>&1
else
  echo "###   BACKUP DRIVE[$disk] ready as $mnt." >> $log 2>&1
  # backup.lst.* contains list of specific backup operations

  while read line;
    do
      # ignore comments lines...
      line=`echo "$line" |sed "s/[ ^I]*#.*//"`
      if [ -z "$line" ]; then continue; fi
      # variable substitutions...
      line=`echo $line | envsubst`
      if ! [[ $line =~ "echo" ]]; then
        echo "EXECUTING '$line' ..." >> $log 2>&1
        fi
        `$line >> $log 2>&1`
      printf "\n\n" >> $log 2>&1
```

```
    done < $flist

  fi


e2=`date +%s`
echo "### BACKUP completed $d2" >> $log 2>&1
elapsed=`expr $e2 - $e`

echo "### ELAPSED TIME: $(date -d "1970-01-01 $elapsed sec" +%H:%M:%S)" >> $log 2>&1

#  save log....
cp $log $mnt/$list.log

# unmount offline backup area...
if [ $mounted -eq 1 ]; then
  # umount drive
  ls -l $mnt >> $log 2>&1
  sleep 2
  umount $mnt
  echo "### UMOUNTED DRIVE $disk" >> $log 2>&1
  fi

#cleanup tmp files
find $logRoot* -mtime +30 | 2>/dev/null xargs -r rm -- >> $log 2>&1

# report log...
cp $log $logRoot
cat $log | /usr/local/bin/mailto.py "BACKUP[$host] -> $list"
```

/usr/local/bin/backup.lst.daily

Example backup list executed by ***/usr/local/bin/backup script***. Shows how to call a list of ***rsync*** calls that perform the backup. Note the variable substitutions defined in /usr/local/bin/backup, Other commands may be called as well, such as the example ***echo***.

```
#!/bin/bash

# updated: 20170228 dvc

# syntax: rsync options source destination
# common rsync options
#       a = rlptgoD
#         recursive, links, permissions, times, groups, owner, devices.
#       v -> verbose
#       z -> compress transfer
#       R -> relative paths
#       -no- -> turn off any options

echo "DAILY BACK UP SCRIPT..."

# system configuration files...
rsync -avzR /etc $backup/daily/$day/

# custom local scripts and cnfiguration...
rsync -avzR /usr/local/bin $backup/daily/$day/
rsync -avzR /usr/local/bin $backup/daily/$day/
rsync -avzR /usr/local/etc $backup/daily/$day/

# web code and databases...
rsync -avzR /home/js/bin $backup/daily/$day/
rsync -avzR /home/js/logs $backup/daily/$day/
```

```
rsync -avzR /home/js/restricted $backup/daily/$day/
rsync -avzR /home/js/sites $backup/daily/$day/
rsync -avzR /home/js/change.log $backup/daily/$day/

echo "... BACKUP COMPLETE"
```

/usr/local/bin/backup_rpi

This script does a backup of the sections of the boot (/boot) and root (/) partitions of the Raspberry Pi main SD card to a "clone" SD card. It assumes a NOOBS setup and partitions on the clone card as built from an existing image. *See SD Card Backup* section for details.

```
#!/bin/bash

# Raspberry Pi SD Card backup script...

start=`date +%s`

# define default paramaters -- must be customized per installation!!!
# Assumes a usb mounted SD card with Raspian installed for backup as...
clone="usb-Generic_Mass-Storage-0:0"
# with two mount points...
clone_mnts="/mnt/os_clone"
boot="/mnt/os_clone/boot"
root="/mnt/os_clone/root"

# define default paramaters
e=`date +_%Y%m%d_%H%M%S`
logRoot="/tmp/backup_rpi"
logBase="$logRoot.log"
log="$logRoot$e.log"


### backup procedure...
echo "### [$HOSTNAME] SC CARD BACKUP BEGUN: `date`" > $log 2>&1
echo "# CLONE:  $clone" >> $log 2>&1
echo "# LOG:    $log" >> $log 2>&1
echo "# MOUNTS: $clone_mnts" >> $log 2>&1
echo "#         $boot" >> $log 2>&1
echo "#         $root" >> $log 2>&1
echo $0 >> $log 2>&1
echo >> $log 2>&1

# check mountpoints exist
if [ ! -d $clone_mnts ]
then
  mkdir $clone_mnts
  echo clone mountpoint $clone_mnts created!
fi
if [ ! -d $boot ]
then
  mkdir $boot
  echo boot clone mountpoint $boot created!
fi
if [ ! -d $root ]
then
  mkdir $root
  echo boot clone mountpoint $root created!
fi

# mount the boot and root partitions of the clone, assumes NOOB partitions
echo "mounting clone partitions..." >> $log 2>&1
mount /dev/disk/by-id/$clone-part6 $boot >> $log 2>&1
```

```
mount /dev/disk/by-id/$clone-part7 $root >> $log 2>&1
echo >> $log 2>&1

# mirror the /boot partition of the main sd card to the clone partition
echo "mirroring /boot partition..." >> $log 2>&1
rsync -aHv --delete /boot/* $boot/ >> $log 2>&1
echo >> $log 2>&1

# mirror the / (root) partition of the main sd card to the clone partition
# excluding special system folders: /boot, /dev, ....
echo "mirroring /, i.e. root partition, excluding special folders..." >> $log 2>&1
echo >> $log 2>&1
# declare a list of root folders to save, excluding special system areas
declare -a folders=("bin" "etc" "home" "lib" "man" "opt" "root" "sbin" "srv" "usr"
"var")
# loop through the folders list
for d in "${folders[@]}"
do
   echo "# mirroring $d..." >> $log 2>&1
   rsync -aHv --delete /$d/ $root/$d/ >> $log 2>&1
   echo >> $log 2>&1
done
echo "checking root symbolic links..." >> $log 2>&1
for s in $( find / -maxdepth 1 -type l ); do
    if [ ! -e $root$s ]; then
        echo "creating symbolic link: $root$s" >> $log 2>&1
        ln -s $(readlink -f $s) $root$s
    fi
done
echo >> $log 2>&1


# unmount partitions...
echo >> $log 2>&1
echo "unmounting partitions..." >> $log 2>&1
umount $boot >> $log 2>&1
umount $root >> $log 2>&1
echo >> $log 2>&1

#cleanup tmp files
find $logRoot* -mtime +30 | 2>/dev/null xargs -r rm -- >> $log 2>&1

stop=`date +%s`
elaspsed=`expr $stop - $start`
echo "### ELAPSED TIME: $(date -d "1970-01-01 $elapsed sec" +%H:%M:%S)" >> $log 2>&1

# report log...
cp $log $logBase
cat $log | /usr/local/bin/mailto.py "SD CARD BACKUP[$HOSTNAME]"
```

/usr/local/bin/backup_img

Script to make an image of the cloned drive. Note: source drive and mount point specific to this setup and may differ for other setups.

```
#!/bin/bash
# makes a backup img of the clone disk
# disk IDs specific to tiny

start=`date +%s`
d=`date +%Y%m%d`
host=`cat /etc/hostname`
src=/dev/sdb
```

```
name=$host-$d
img=/mnt/usb/img/$name.gz
log=/tmp/backup_img-$name.log

echo "### BACKUP IMAGE CREATION BEGUN: `date`" > $log 2>&1
echo "# HOST:   $host" >> $log 2>&1
echo "# SOURCE: $src" >> $log 2>&1
echo "# IMAGE:  $img" >> $log 2>&1
echo "# LOG:    $log" >> $log 2>&1
echo  >> $log 2>&1

# image creation...
dd if=$src bs=4K | gzip > $img

stop=`date +%s`
elapsed=`expr $stop - $start`
echo "### ELAPSED TIME: $(date -d "1970-01-01 $elapsed sec" +%H:%M:%S)" >> $log 2>&1

cat $log | /usr/local/bin/mailto.py "BACKUP_IMG[$name]"
```

/usr/local/bin/bsync

Single script that integrates all the backup actions of ***/usr/local/backup***, ***/usr/local/backup_rpi*** and, ***/usr/local/backup_img***, while using a single configuration file ***/usr/local/etc/bsync.conf***.

```
#!/bin/bash
#/usr/local/bin/bsync

# generic script for performing backup/sync based on command scripts ...
#   1. Performs general file backup from source to destination
#   2. Or copies boot and root OS partions to clone disk
#   3. Or makes a clone image

# default configuration
dflt="/usr/local/etc/bsync.conf"

# SYNTAX:
if [[ $1 =~ '-h' ]]; then
  echo "bsync -h|script_tag [configuration_file]"
  echo "  default configuration: $dflt"
  exit
fi

# links to configuration data file ...
export data=${$2:-$dflt}

# functions...
export db=0     # debug flag: -1: stdout, 0: logfile, 1: logfile & stdout
#debug echo...
dbg () {
  if [[ $db -ne -1 ]]; then echo $* >> $log 2>&1; fi
  if [[ $db -ne 0 ]]; then echo $*; fi
}
# debug execute...
dbgx () {
##  echo "LINE: $*"
  case $db in
    -1)
      $* 2>&1
      ;;
    0)
      $* 2>&1 >> $log
      ;;
```

```
    1)
      $* 2>&1 | tee -a $log
      ;;
  esac
}

# script variables...
#   exported variables and environment variables can be substituted
#   in sync scripts configurations and commands
export abort=0
# tag identifies specific sections in configuration/command data file.
export tag=${1:-test} # default to test, if not specified

#create a unique log filename...
e=`date +%s`
logRoot=/tmp/bsync-$tag.log
export log="$logRoot.$e"
touch $log
dbg "Log: $log"

# generate date info...
export day=`date +%A`
export mday=`date +%d`
export week=$((10#`date +%W` % 4))
export month=`date +%B`
export year=`date +%Y`
if [[ $((10#`date +%W` % 2)) -eq 0 ]];
  then export wk="even";
  else export wk="odd";
fi
if [[ $((10#`date +%m` % 2)) -eq 0 ]];
  then export mon="even";
  else export mon="odd";
fi


# SCRIPT ACTION BEGINS HERE...
# source default and specified backup configurations variable declarations...
for cfg in "dflt-cfg" "$tag-cfg"
  do
    dbg "# reading $cfg configuration..."
    # parse section from cfg file and source as part of local script
    source <(nawk "/^\[$cfg\]/{flg=1;next}/^\[/{flg=0}flg" $data)
  done

# setup script stuff...
title=$( echo "### [$HOSTNAME] $tag SYNC BEGUN..." | tr a-z A-Z )
dbg $title
dbg "# DATE: Week $week [$wk] - $day, $mday $month [$mon] $year"
dbg

# report disk status...
dbg "PRE-RUN disk status..."
dbgx "df -B M --total -x tmpfs -x devtmpfs"
dbg

# if offline destination drive, mount it first...
# requires $dev, $mnt, and $mount definitions in configuration section!
if [[ $mount -eq 1 ]]; then
  try=6
  n=0
  while [ $n -lt $try ];
```

```
    do
      # mount destination drive and give it a moment then test if ready
      dbgx mount -t ext4 $dev $mnt
      sleep 1
      if ! grep -qs "$mnt" /etc/mtab
      then
        # failed to mount, retry
        (( n = $n + 1 ))
        dbg "###   BACKUP DRIVE[$dev] failed to mount on attempt $n."
        sleep 9
        if [[ $n -eq $try ]]; then
          dbg "### BACKUP DRIVE[$dev] NOT READY: BACKUP ABORTED!"
          abort=1
        fi
      else
        # good to go
        dbg "###   BACKUP DRIVE[$dev] successfully mounted as $mnt."
        ls -l $mnt >> $log 2>&1
        break
      fi
    done
fi

# PERFORM BACKUP COMMANDS...
# requires $mode definition in configuration!
# requires $dryrun definition in configuration!
if [[ $abort -eq 0 ]]; then
  case $mode in
    image)
      # create image from cloned OS device
      # image destination file
      img_name=$HOSTNAME-$e.img
      img=$dest/$img_name.gz
      # log info...
      dbg "### $host IMAGE CREATION BEGUN: `date`"
      dbg "# SOURCE: $src --> DEST: $img"
      dbg "# LOG:    $log"
      # image creation...
      dd if=$src bs=4K | gzip > $img
      subj=`echo "$HOSTNAME IMG CREATION: $name" | tr a-z A-Z`
      ;;
    clone)
      if [ $os = 'raspian' ]; then
        bpart='part1'
        rpart='part2'
      else
        bpart='part6'
        rpart='part7'
      fi
      ### clone procedure...
      dbg "### $( echo [$HOSTNAME] $os | tr a-z A-Z ) CLONE BEGUN: `date`"
      dbg "# LOG:    $log"
      dbg "# CLONE: $dev"
      dbg "# MOUNT: $mnt"
      dbg "#   ROOT:  $rpart --> $mnt"
      dbg "#   BOOT:  $bpart --> $mnt/boot"
      dbg
      # check mountpoint exist or create...
      if [ ! -d $mnt ]; then
        mkdir -p $mnt
        dbg "clone mount $mnt created!"
      fi
```

```
      # mount the boot and root partitions of the clone
      dbg "mounting clone root partition..."
      dbgx mount $dev-$rpart $mnt
      if [ ! -d $mnt/boot ]; then
        mkdir $mnt/boot
        dbg "clone boot mount $mnt/boot created!"
      fi
      dbg "mounting clone boot partition under clone root..."
      dbgx mount $dev-$bpart $mnt/boot
      # mirror / (root) of system to the clone root partition,
      # excluding special system folders: /proc, /dev, ....
      dbg "mirroring / (i.e. root) to clone device at $mnt ..."
      # loop through the root_folders list...
      for d in "${root_folders[@]}"
        do
          dbg "  mirroring folder /$d to $mnt/$d..."
          dbgx rsync -aHv --delete /$d/ $mnt/$d/
        done
      dbg "checking root symbolic links..."
      for s in $( find / -maxdepth 1 -type l )
        do
          if [ ! -e $mnt$s ]; then
            dbg "  creating symbolic link: $mnt$s"
            ln -s $(readlink -f $s) $mnt$s
          fi
        done
      # report disk status...
      dbg "POST CLONE disk status..."
      dbgx "df -B M --total -x tmpfs -x devtmpfs"
      dbg
      # unmount partitions...
      dbg "unmounting partitions..."
      dbgx umount $mnt/boot
      dbgx umount $mnt
      subj=$( echo "$HOSTNAME $os CLONE" | tr a-z A-Z )
      ;;
    backup|*)
      # requires $src and $dest definitions in configuration section!
      dbg "###   BACKUP $src to $dest..."
      # command script contains list of specific backup operations
      while read line;
        do
          # ignore comments lines...
          line=`echo "$line" | sed "s/[ ^I]*#.*//"`
          if [ -z "$line" ]; then continue; fi
          # variable substitutions...
          line=`echo "$line" | envsubst`
          if [[ $dryrun -eq 1 ]]; then line=`echo "$line" | sed "s/rsync/rsync -n/"`;
fi
          if ! [[ $line =~ "echo" ]]; then
            dbg "EXECUTING '$line' ..."
          fi
          dbgx "$line"
        done < <(nawk "/^\[$tag\]/{flg=1;next}/^\[/{flg=0}flg" $data)
      dbg
      dbg
      # report disk status...
      dbg "POST-BACKUP disk status..."
      dbgx "df -B M --total -x tmpfs -x devtmpfs"
      dbg
      subj=$( echo $HOSTNAME $desc BACKUP | tr a-z A-Z )
      ;;
```

```
  esac
fi


# CLEANUP...
# compute and report elasped time...
elapsed=$(date -d "1970-01-01 `expr $(date +%s) - $e` sec" +%H:%M:%S)
dbg "### BACKUP COMPLETED! ELAPSED TIME: $elapsed"

# unmount offline backup area if defined...
if [[ $mount -eq 1 ]]; then
  # umount drive
  ls -l $mnt >> $log 2>&1
  sleep 2
  umount $mnt >> $log 2>&1
  mx=`mount | grep "$mnt" | awk 'NR==1{print $1}'`
  if [[ ! -z $mx ]]; then
    dbg "ERROR: $mnt [$mx] FAILED TO UNMOUNT!"
  else
    dbg "### SUCCESSFULLY UMOUNTED DRIVE $dev"
  fi
fi

#cleanup older tmp files
dbgx "find $logRoot* -mtime +30 -delete"

# report log...
cp $log $logRoot
cat $log | /usr/local/bin/mailto.py "$subj"
```

/usr/local/etc/bsync.conf

Configuration data example for ***/usr/local/bin/bsync***.

```
#!/bin/bash
#/usr/local/etc/bsync.conf

# created:  20171211 dvc
# released: 20171217 dvc

# sections define configuration or command scripts for bsync script...
#   bsync <tag> [<configuration_file>]
# each configuration section begins with [<tag>-cfg]
#   configuration sections sourced to main script (i.e. should be exports)
# each command script section begins with [<tag>]
#   command sections should execute specific backup commands
# a new section or end-of-file terminates the previous section
# order does not matter, but organized alphabetically

# customized configurations for backup actions...
# OS cloning configuration...
[clone-cfg]
export desc="clone"
export dev="/dev/disk/by-id/usb-Generic_STORAGE_DEVICE_000000000272-0:0"
export mnt="/mnt/clone"
# define the specific partitioning setup: NOOBS or Raspian
#export os="noobs"
export os="raspian"
# declare a list of root folders to save, excluding special system areas
declare -a root_folders=("bin" "boot" "etc" "home" "lib" "opt" "root" "sbin" "srv"
"usr" "var")
export mode="clone"
```

```
# daily specific configuration...
[daily-cfg]
# assume defaults
export desc="DAILY [$day]"

# datos-to-safe backup configuration
[datos-cfg]
export desc="datos"
export dev="/dev/disk/by-label/safe"
export mnt="/mnt/safe"
export src="/mnt/datos"
export dest="$mnt/datos"
export mount=1

# configuration defaults...
[dflt-cfg]
export desc="unknown"
export src="/"
export dest="/mnt/backup/$HOSTNAME"
export mode="backup"
export mount=0
export dryrun=0

[img-cfg]
# image creation configuration
export desc="image"
export src="/dev/disk/by-id/usb-Generic_STORAGE_DEVICE_000000000272-0:0"
export mnt="/mnt/datos"
export dest="$mnt/source/Linux/Raspian"
export mode="image"

# monthly specific configuration...
[monthly-cfg]
# assume defaults
export desc="MONTHLY [$mon]"

# a test configuration...
[test-cfg]
export desc="test"
export test="check"
export mode="test"

# weekly specific configuration...
[weekly-cfg]
# assume defaults
export desc="WEEKLY [$wk]"

[end-of-cfg]
# dummy marker to terminate configuration sections


# backup command scripts...
#   read line-by-line
#   NOTE: bash export builtin doesn't work here!
# rsync options
#    a = rlptgoD
#    recursive, links, permissions, times, groups, owner, devices.
#    v -> verbose
#    z -> compress transfer
#    R -> relative paths
#    -no- -> turn off any options
#       n, --dry-run to show but prohibit action
```

```
[daily]
echo "$desc BACKUP SCRIPT..."

# system configuration files...
rsync -avzR /etc $dest/daily/$day/

# custom local scripts and cnfiguration...
rsync -avzR /usr/local/bin $dest/daily/$day/
rsync -avzR /usr/local/etc $dest/daily/$day/
rsync -avzR /usr/local/samba $dest/daily/$day/

# home folders
rsync -avzR /home $dest/daily/$day/

# web code and databases...
#rsync -avzR /home/js/bin $dest/daily/$day/
#rsync -avzR /home/js/logs $dest/daily/$day/
#rsync -avzR /home/js/restricted $dest/daily/$day/
#rsync -avzR /home/js/sites $dest/daily/$day/
#rsync -avzR /home/js/change.log $dest/daily/$day/

echo "... BACKUP COMPLETE"


[datos]
echo "HOME FILE SERVER BACKUP SCRIPT..."

# personal folders...
rsync -avz --delete $src/archived $dest
rsync -avz --delete $src/home $dest
rsync -avz --delete $src/webs $dest
rsync -avz --delete $src/Saranam $dest

#media/folders...
rsync -avz --delete $src/media/magazines $dest/media
rsync -avz --delete $src/media/music $dest/media
rsync -avz --delete --exclude family/Baby/ --exclude family/Patrick/Travel/ --exclude
Thumbs.db $src/media/pictures $dest/media/

#find . -name "Thumbs.db" -delete

echo "... BACKUP COMPLETE"


[monthly]
echo "$desc BACKUP SCRIPT..."

# system configuration files...
rsync -avzR /etc $dest/monthly/$mon/

# custom local scripts and cnfiguration...
rsync -avzR /usr/local/bin $dest/monthly/$mon/
rsync -avzR /usr/local/etc $dest/monthly/$mon/
rsync -avzR /usr/local/samba $dest/monthly/$mon/

# home folders
rsync -avzR /home $dest/monthly/$mon/

# web code and databases...
#rsync -avzR /home/js/bin $dest/monthly/$mon/
#rsync -avzR /home/js/logs $dest/monthly/$mon/
```

```
#rsync -avzR /home/js/restricted $dest/monthly/$mon/
#rsync -avzR /home/js/sites $dest/monthly/$mon/
#rsync -avzR /home/js/change.log $dest/monthly/$mon/

echo "... BACKUP COMPLETE"


[test]
echo "TEST BACKUP SCRIPT..."
# test script...
echo SOURCE: $src ==> DESTINATION: $dest
ls /tmp/backup*
echo "... BACKUP COMPLETE"


[weekly]
echo "$desc BACKUP SCRIPT..."

# system configuration files...
rsync -avzR /etc $dest/weekly/$week/

# custom local scripts and cnfiguration...
rsync -avzR /usr/local/bin $dest/weekly/$week/
rsync -avzR /usr/local/etc $dest/weekly/$week/
rsync -avzR /usr/local/samba $dest/weekly/$week/

# home folders
rsync -avzR /home $dest/weekly/$week/

# web code and databases...
#rsync -avzR /home/js/bin $dest/weekly/$week/
#rsync -avzR /home/js/logs $dest/weekly/$week/
#rsync -avzR /home/js/restricted $dest/weekly/$week/
#rsync -avzR /home/js/sites $dest/weekly/$week/
#rsync -avzR /home/js/change.log $dest/weekly/$week/


echo "... BACKUP COMPLETE"
```

/usr/local/bin/croncall

Wrapper script for calling scripts form cron that simply logs the action, mainly for debug.

```
#!/bin/bash
# wrapper script to call and log cron jobs

d=`date`
log='/tmp/croncall.log'
vlog='/var/log/croncall.log'

echo "[$d] $*" >> $log 2>&1

# set a working path since cron does not run in a shell...
PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:$PATH"
#echo "  PATH: $PATH" >> $log 2>&1

# save a truncated version (as tmp clears on power cycle)
tail -n 40 $log > $vlog

# call the task script with any parameters...
# $* = $1 ...
$*
```

/usr/local/bin/gotmux

Shortcut “GOT MUX” (or GO TMUX) command that attaches a user to their predefined tmux session with $USER session name. See *~/.tmux.init* for info on starting a user specific session.

```
#!/bin/bash
# start user specific tmux session

who=${1-$USER}
echo "WHO: '$who'"

if [ "$who" == "$USER" ]; then
    tmux attach-session -t $who
else
    su $who -c "tmux attach-session -t $who"
fi
```

/usr/local/bin/mailto.py

Script called from other scripts that emails log files and other information based on system actions.

Be sure to change the user and server specific information below highlighted in red.

```
#!/usr/bin/python

import sys
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

# compose fields...
subject = "SARANAM CASITA SCRIPT: " + sys.argv[1]
if len(sys.argv)>2:
  you = ','.join(sys.argv[2:])
else:
  you = "email-address"
me = "email-address"
server = "mail.noip.com"
port = "465"
username = "username"
pw = "**********"
message = ''
for line in sys.stdin:
  message += line

# Create message container - the correct MIME type is multipart/alternative.
msg = MIMEMultipart('alternative')
msg['Subject'] = subject
msg['From'] = me
msg['To'] = you
# Create the body of the message (a plain-text and an HTML version).
text = "Automatic script ..."
html = """\
<html>
  <head></head>
  <body>
    <p>%s</p>
    <pre>%s</pre>
  </body>
</html>
""" % (subject,message)

# Record the MIME types of both parts - text/plain and text/html.
part1 = MIMEText(text, 'plain')
```

```
part2 = MIMEText(html, 'html')

# Attach parts into message container.
# According to RFC 2046, the last part of a multipart message, in this case
# the HTML message, is best and preferred.
msg.attach(part1)
msg.attach(part2)

# Send the message via local SMTP server.
s = smtplib.SMTP_SSL(server,port)
s.login(username,pw)
# sendmail function takes 3 arguments: sender's address, recipient's address
# and message to send - here it is sent as one string.
s.sendmail(me, you, msg.as_string())
s.quit()
```

/etc/init.d/noip2

Script to start and stop No-IP Dynamic Update Client (DUC) daemon.

```
#! /bin/sh
# /etc/init.d/noip2
# Supplied by no-ip.com

### BEGIN INIT INFO
# Provides:     noip2
# Required-Start: networking
# Required-Stop:
# Should-Start:
# Should-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Start noip2 at boot time
# Description: Start noip2 at boot time
### END INIT INFO

DAEMON=/usr/local/bin/noip2
NAME=noip2

test -x $DAEMON || exit 0

case "$1" in
    start)
    echo -n "Starting dynamic address update: "
    start-stop-daemon --start --exec $DAEMON
    echo "noip2."
    ;;
    stop)
    echo -n "Shutting down dynamic address update:"
    start-stop-daemon --stop --oknodo --retry 30 --exec $DAEMON
    echo "noip2."
    ;;
    restart)
    echo -n "Restarting dynamic address update: "
    start-stop-daemon --stop --oknodo --retry 30 --exec $DAEMON
    start-stop-daemon --start --exec $DAEMON
    echo "noip2."
    ;;
    status)
    echo -n "NOIP DUC status..."
    $DAEMON -S
    ;;
    *)
```

```
    echo "Usage: $0 {start|stop|restart|status}"
    exit 1
esac
exit 0
```

Note: For setup run noip2 –C ***to configure the client, which saves the configuration to /usr/local/etc/no-ip2.ocnf. Use*** noip2 –S ***to show the configuration data.***

/usr/local/bin/pid

Utility script to find running process by a given identifier string. For example ***pid noip*** will show any running No-IP DUC services.

```
#!/bin/bash
# list processes by name
report=$(ps aux | sed -n -e 1p -e "/$1/I"p | grep -v $0 | grep -v sed)
echo "$report"
```

/usr/local/bin/samba

Init script for samba.

```
#!/bin/bash

# script to setup samba (AD DC)
# can't get /etc/init.d/samba-ad-dc to run ???


### BEGIN INIT INFO
# Provides:  samba
# Required-Start:
# Required-Stop:
# Should-Start:
# Should-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Start and Stop Samba
# Description:  samba initialization script
### END INIT INFO

# parameters...
NAME=samba
PROMPT="${NAME} Version 4"
SCRIPT=/usr/local/samba/sbin/samba
KILL="killall"
RC=/usr/sbin/update-rc.d


case "$1" in
        start)
                echo "Starting $PROMPT ($SCRIPT) service..."
                $SCRIPT
                sleep 3
                $0 status
        ;;
        stop)
                echo "Shutting down $PROMPT service..."
                $KILL $NAME
                sleep 3
                $0 status
        ;;
        status)
```

```
                STATUS=$(ps auxf | sed -n -e 1p -e "/$NAME/I"p | grep -v $0 | grep -v
sed)
                echo "$STATUS"
        ;;
        restart)
                $0 stop
                sleep 1
                $0 start
        ;;
        auto)
                case "$2" in
                        start)
                                $RC $NAME defaults
                        ;;
                        stop)
                                $RC $NAME remove
                        ;;
                        *)
                                $0
                        ;;
                esac
        ;;
        *)
                echo "Usage: $0 start|stop|status|restart|(auto start|auto)|stop"
                exit 1
        ;;
esac
```

/usr/local/bin/strip

Useful little script to strip and report only meaningful lines from configuration files.

```
#!/bin/bash
# strip comments and blank lines from configuration files.
# Usage: strip <configuration_file>
grep "^[^#;*].*$" $1
```

/etc/init.d/tmux

Script to launch tmux for each specific user having a ***.tmux.init*** file in their home directory.

```
#!/bin/bash

# script to setup terminal multiplexer with session for multiusers
# each user defines their setup in /home/$USER/.tmux.init
# session named for user

### BEGIN INIT INFO
# Provides:  tmux-$USER
# Required-Start:
# Required-Stop:
# Should-Start:
# Should-Stop:
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Start and Stop
# Description:  tmux initialization script
### END INIT INFO

# parameters...
NAME=tmux
SCRIPT=/usr/local/bin/tmux
XARGS="kill-server"
```

```
# find users  with .tmux.init
declare -a who
for u in $( ls /home ); do
    if [ -e /home/$u/.tmux.init ]; then
        who[${#who}]=$u
    fi
done

case "$1" in
        start)
                echo "Starting $SCRIPT service..."
                for u in $who; do
                        su $u -c "$SCRIPT -c /home/$u/.tmux.init"
                        echo "Started: /home/$u/.tmux.init"
                done
                sleep 1
                $0 status
        ;;
        stop)
                echo "Shutting down $SCRIPT service..."
                su $USER -c "$SCRIPT $XARGS"
                $0 status
        ;;
        status)
                STATUS=$(ps aux | sed -n -e 1p -e "/$NAME/I"p | grep -v $0 | grep -v
sed)
                echo "$STATUS"
        ;;
        restart)
                $0 stop; $0 start
        ;;
        auto)
                case "$2" in
                        start)
                                /usr/sbin/update-rc.d $NAME defaults
                        ;;
                        stop)
                                /usr/sbin/update-rc.d $NAME remove
                        ;;
                        *)
                                $0
                        ;;
                esac
        ;;
        *)
                echo "Usage: $0 start|stop|status|restart[|auto start|auto stop"
                exit 1
        ;;
esac
```

/usr/local/bin/update

Script to automatically make server OS updates and notify admin of action, errors, and pending reboot. Note: Accesses /etc/motd (message of the day shown at login) to query reboot.

```
#!/bin/bash
# script to make OS updates

e=`date +%s`
d=`date`
logRoot='/tmp/update'
tmp="$logRoot.tmp"
log="$logRoot.$e"
```

```
host=`cat /etc/hostname`

echo "$0 RUN $d with $PATH" > $log 2>&1
echo "" >> $log 2>&1
# run system update...
apt-get update -y > $tmp 2>&1
apt-get upgrade -y >> $tmp 2>&1
echo "" >> $tmp 2>&1
echo "" >> $tmp 2>&1

# remove logs over 30 days old...
find $logRoot.* -mtime +30 | 2>/dev/null xargs -r rm -- >> $tmp 2>&1
echo "" >> $tmp 2>&1

# is restart required...
msg=$(grep "restart required" /etc/motd)
check=$(grep -c "restart required" /etc/motd)
echo "MSG: $msg" >> $log 2>&1
if [ $check -ne 0 ]
then
  echo "REBOOT" >> $log 2>&1
else
  echo "OK" >> $log 2>&1
fi
echo "" >> $log 2>&1

echo "Errors?..." >> $log 2>&1
grep -i "error" $tmp >> $log 2>&1
echo "-- END OF ERRORS --" >> $log 2>&1
echo "" >> $log 2>&1

echo "Update Log..." >> $log 2>&1
cat $tmp >> $log
cp $log "$logRoot.log"
cat $log | /usr/local/bin/mailto.py "$host update script..."
```

/usr/local/bin/ups

Script called from apcupsd daemon to notify admin of UPS state changes such as lost power. Links must be added to /etc/apcupsd folder to point to this script for each desired power state warning. For example

```
sudo ln –s /usr/local/bin/ups /etc/apcupsd/doshutdown
```

This causes an ***apcupsd doshutdown*** event to call ***/usr/local/bin/ups***. Other states include ***mainsback***, ***offbattery***, ***onbattery***, ***powerout***, etc. See ***man apcupsd*** for details.

```
#!/bin/bash

# script to report UPS state changes...

e=`date +%s`
d=`date`
logRoot='/var/log/ups'
log="$logRoot.$e"
host=`cat /etc/hostname`

echo "$0 $* RUN $d" > $log 2>&1
echo "" >> $log 2>&1

# dump apcupsd log...
tail /var/log/apcupsd.events >> $log 2>&1
```

```
echo "" >> $log 2>&1

# remove logs over 30 days old...
find $logRoot.* -mtime +30 | 2>/dev/null xargs -r rm -- >> $log 2>&1

# move log to base file and email to sysop...
cp $log "$logRoot.log"
cat $log | /usr/local/bin/mailto.py "$host UPS script..."
```

/etc/init.d/vncserver

Script to control operation of tightvncserver.

```
#!/bin/sh
### BEGIN INIT INFO
# Provides: vncboot
# Required-Start: $remote_fs $syslog
# Required-Stop: $remote_fs $syslog
# Default-Start: 2 3 4 5
# Default-Stop: 0 1 6
# Short-Description: Start VNC Server at boot time
# Description: Start VNC Server at boot time.
### END INIT INFO

USER=root
HOME=/root

export USER HOME

case "$1" in
 start)
   echo "Starting VNC Server"
   #Insert your favoured settings for a VNC session
   /usr/bin/vncserver :0 -geometry 1280x800 -depth 16 -pixelformat rgb565
   ;;

 stop)
   echo "Stopping VNC Server"
   /usr/bin/vncserver -kill :0
   ;;

 *)
   echo "Usage: /etc/init.d/vncboot {start|stop}"
   exit 1
   ;;
esac

exit 0
```

Appendix R: Terminal Multiplexer (tmux), Install, Setup, and Use

Terminal multiplexer creates a nice environment for multi-terminal remote server operation. You can configure it to start automatically after login and users can highly customize the operating experience.

To install, run the following, which installs tmux in /usr/local/bin

```
sudo apt-get install tmux
```

Run terminal multiplexer with a predefined window/pane configuration (see ***.tmux.init***) located in the user's home folder such as

```
tmux –c ~/.tmux.init
```

Once started a tmux session will run indefinitely. To reconnect after logging out and back in run

```
tmux attach
```

or if multiple sessions are running run the following:

```
tmux attach-session –t <session_name>
```

~/.tmux.init

This defines an example shell-command file called by tmux that establishes a tmux session with a default set of windows running various processes. The user can easily modify the number of windows, names, and purpose of each window by editing this file, which uses BASH syntax.

```
#!/bin/bash

# preloaded session...
/usr/local/bin/tmux new-session -d -s $USER


# define a set of windows and launch specific commands
# to make windows remain if initial command terminates append a "bash -i" command

# web server
/usr/local/bin/tmux new-window -c /home/js/bin -n server '/usr/bin/nodejs ./homebrew.js
../restricted/config.json; bash -i'

# interactive node shell
/usr/local/bin/tmux new-window -c /home/js/bin -n node-bin '/usr/bin/nodejs; bash -i'

# interactive node shell
/usr/local/bin/tmux new-window -c /tmp -n node-tmp '/usr/bin/nodejs; bash -i'

# general purpose bash shell
/usr/local/bin/tmux new-window -c /home/js/bin -n bash-bin

# general purpose bash shell
/usr/local/bin/tmux new-window -c /home/js -n bash-home

# interactive python window
/usr/local/bin/tmux new-window -c /home/js/bin -n python-bin '/usr/bin/python; bash -i'
```

~/.tmux.conf

The ~/.tmux.conf file allows each user to uniquely configure tmux to their preferred style. Configuration options exceed the scope of this document but the example below provides a starting point. See tmux documentation for more information.

This configuration:

- Redefines the hot-key “prefix” as CTRL-A.
- Binds the PREFIX-r key to reload the configuration file.
- Sets some basic window appearance features: titles, white text on black, …
- Defines some mouse defaults.

```
# tmux configuration 20140522...

# change the prefix hot-key to more useable CTRL-A
set -g prefix C-a
unbind C-b
bind C-a send-prefix

# binding for the config file
bind r source-file ~/.tmux.conf

# window look and feel...
set -g set-titles on
set -g status-utf8 on
set -g status-fg black
set -g status-bg white
set -g status-interval 5

# enable mouse to allow window scrolling...
set-window-option -g mode-mouse on
set-option -g mouse-select-pane on
set-option -g mouse-resize-pane on
set-option -g mouse-select-window on

# toggle mouse-mode with prefix m or prefix M...
bind m \
  set -g mode-mouse on \;\
  set -g mouse-resize-pane on \;\
  set -g mouse-select-pane on \;\
  set -g mouse-select-pane on \;\
  set -g mouse-select-window on \;\
  display 'Mouse: ON'
bind M \
  set -g mode-mouse off \;\
  set -g mouse-resize-pane off \;\
  set -g mouse-select-pane off \;\
  set -g mouse-select-pane off \;\
  set -g mouse-select-window off \;\
  display 'Mouse: OFF'
```

Appendix S: Building Samba or Other Packages

Building Samba or any Linux system code follows a relatively straightforward process, which is the same flow for any package just deviating in the package details and dependencies.

Create a temporary folder (under user pi) for the build.

```
mkdir samba
cd samba
```

Before building Samba, first install its dependencies. The list may change from version to version but should include …

```
sudo apt-get -y install dnsutils
sudo apt-get -y install acl
sudo apt-get -y install attr
sudo apt-get -y install krb5-user
sudo apt-get -y install python-dev
sudo apt-get -y install libgnutls28-dev
sudo apt-get -y install libgnutlsxx28
sudo apt-get –y install libldap2-dev
sudo apt-get -y install cups
```

> ***Tip: You can install multiple packages in a single command, but if one fails it likely will not install any. Doing one at a time is a little easier to watch and debug what is happening.***

> ***Tip: To find missing packages try apt-cache search <package>, where <package> defines a simple name string such as libgnutls.***

Then get the Samba source (~20M) with the following commands. This may take a while to download depending on connection speed.

```
wget https://www.samba.org/samba/ftp/samba-latest.tar.gz
tar vzxf samba-latest.tar.gz
cd samba-x.y.z
```

Now the steps to compile. This will take over 5 hours on a Raspberry Pi B+.

```
./configure  --enable-debug --enable-selftest
make
```

> ***Note: The configure step above could fail based on missing dependencies. If so, simply Google information as to where to find and how to install each missing dependency until configure no longer reports any errors. You can then proceed with the make operation.***

If make completes successfully then install with

```
sudo make install
```

If make install completes successfully, the compiled version of samba should be installed. At this point, exit from the ***su*** shell or continue with other samba configuring operations. See Samba Wiki.

Notes

1. You may remove the build directory at this point.
2. You may also want to add /usr/local/samba to the path variable. *See 0*

3. *See also* Appendix C: Custom Scripts for useful “embellished” commands, such as pid, strip, update.
4. Linux User Tips.
5. Run ***update-rc.d*** command for packages that you wish to start automatically at startup.

Appendix T: Cloning a Disk or SD Card

> ***Tip: The Raspian Desktop GUI now includes a new SD card copier (under Accessories) to backup an SD card, even on a running system, supports both Raspian and NOOBS builds, and can copy to a smaller or larger card as long as the card has enough room for all OS files.***

Cloning a disk or SD card makes an exact copy sector by sector to a single image file (***.img***). Creating an image stores all disk data including free space. To restore the image file requires a disk having an equal or larger number of sectors as the original card.

> ***Warning: All SD cards of the same size do not have the same capacity, that is, you may not be able to restore an 8G image to an 8G card if it has slightly less sectors.***

1. Shut down or halt the Pi and remove the SD card.
   (You cannot reliably copy the SD card while mounted as the root file system.)
2. Insert the SD card into your computer.
3. For Windows …
   a. Start up Win32DiskImager.
   b. In the "Image File" box, enter the path for the image file being created, such as ***C:\data\images\tiny.img***
   c. Under the "Device" box, select the disk representing the SD card.
   d. Click the "Read" button to create the image file from the card.
   e. When done, eject the SD card.

   Use this process in reverse to build an SD card from an image file.

   For Linux …
   a. Mount the SD card as a secondary disk (i.e. non-booting) in another machine.
   b. Use the dd command to copy as:

```
sudo dd if=</dev/sdx> of=</path/to/image>
```

   Where x of sdx is the letter signifying the SD card disk device.

   Or redirect to gzip for compression...

```
dd if=</dev/sdx> | gzip > </path/to/image.gz>
```

   Or a live update to another Linux box…

```
ssh root@raspberrypi dd if=/dev/mmcblk0 | gzip -c > img.gz
```

4. Replace the SD card in the Raspberry Pi and power it back on.

Keep the backup IMG file in a safe place. For immediate server restoration, copy the image to a second backup SD card using Win32DiskImager or Linux dd command in reverse or directly using 7Zip to extract an img file to an SD card.

For non-NOOBS installs, https://github.com/billw2/rpi-clone offers a way to clone Raspberry Pi SD cards in-situ.

The script ***/usr/local/bin/backup_rpi*** keeps a cloned SD card up to date. Run it after periodic updates (i.e. ***/usr/local/bin/update***).

Appendix U: Google Drive

A number of services provide efficient, secure, and online/offsite file storage with backup. This setup assumes the use of ***Google Drive*** because of its direct integration with ***Gmail*** in use by the example installation and other services such as ***Google+***, ***Google Photos***, and ***Chrome***, as well as being low cost for additional storage. Every Google account automatically includes 15G of storage aggregated across ***Gmail*** and all other ***Google Apps***, including ***Drive***.

> ***Tip: See the PowerPoint presentation entitled Using Google Drive for more details.***

Google Drive Web Interface

Google Drive provides a web-interface using the Chrome browser to completely access and manage using the following link:

https://drive.google.com/drive/my-drive

From Chrome, you can simply type drive in the address bar. ***You must log into your Google Account first.*** Through the web interface you can upload, download, move, delete, share, and manage file versions, even on machines without a Google Drive installation.

Drive Installation

Additionally, you can install the ***Drive*** app on a local machine, Windows or Mac or mobile devices, to provide automatic file synchronizing with a local folder. Download ***Drive*** from

https://www.google.com/drive/download/

Simply click on the downloaded file to install. This will create a ***Google Drive folder*** under your user area (i.e. c:\Users\dvc\Google Drive) that will appear in the Quick access list found in the top left pane of the ***Windows Explorer*** window as seen below.

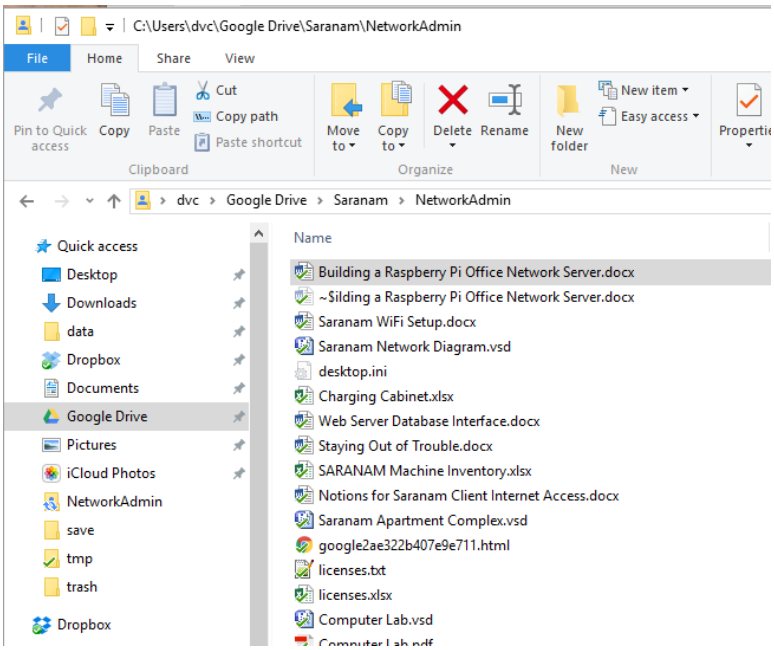

Figure 3: Google Drive Quick Access Link

Documents Backup and Offsite Access

> ***Warning: This will not work for installations using folder redirection for roaming profiles.***

> ***Info: As of approximately October 2017, Google has replaced the Google Drive Desktop App with Backup and Sync, which can be configured to automatically include the local documents folder, making the instructions of this section obsolete.***

A nice feature ***Windows*** and a ***Google Drive*** installation involves the direct mapping of one's Document folder to the Google Drive folder. This provides automatic backup of Documents to Google Drive.

> ***Warning: Before doing this setup I recommend backing up your Documents folder. This can be done my making a temporary copy to c:\temp\Documents for example.***

> ***Tip: For older versions of Windows prior to Windows 10, reference the My Documents folder instead of Documents.***

To setup, in ***Windows Explorer*** click on ***This PC*** in the left-hand pane to expand its subfolders. Write click on the ***Documents*** folder and click on ***Properties...***. In the dialog window select the ***Location tab***. Click ***Move*** and browse for the ***Google Drive*** folder. Select the ***Select Folder*** button. ***Windows*** will move the files in the ***Documents*** folder to the ***Google Drive folder*** and point to that folder for future file operations. ***Drive*** will automatically sync files to the Google account storage.

Selective Copying

Google Drive supports selective copying from your Google account to a local machine (but not vice versa). As such, you can conserve disk space on your local machine but syncing only specific subfolders located within the Google account. To do so right-click on the Google Drive folder in Windows Explorer and select the Google Drive item and then Preferences. Alternately, click on the Google Drive icon in the System Tray on the lower right Taskbar area. Then click the 3 vertical dots in the upper right of the popup and then select Preferences. Then chose the folders you wish to sync with Google Drive.

Common Shared Area with Local Drive Mapping

Drive also has value as a common file share resource. This case uses a separate standalone Gmail account, such as ***shared-gmail-address***. Given the greater storage needs you can purchase additional storage space as needed. Share trees within the common share with users and then each user can "Add to my drive" to have direct access.

Appendix V: Active Directory Users and Computers

Windows ***Remote Server Administration Tools*** (RSAT) provide an interface to the ***Active Directory*** server. The ***Active Directory Users and Computers*** plugin supports managing both domain users and computers. This runs on one or more Windows machines attached to the domain to enable GUI-based management without having to execute commandline commands directly on the server itself.

Adding a Computer

You can add computers to the domain most easily from the specific machine following these steps.

1. Click on the Windows icon (at the lower left of the desktop).
2. Type **join**. Select **Rename your PC or join a domain** from the list of choices.
3. Click **Join a Domain** from the Settings dialog.
   (You must first disconnect from any other domain.)
4. Type **saranam** and click **Next**.
5. Enter admin credentials to authorize the join.

Adding a Group

To add a group follow these steps…

1. Select Users in the left pane under the domain. Right-click and select New → Group.
2. Enter the group name.
3. Use the default global security group settings. Click OK.
4. Click on the created group and open its property dialog and add a description.

Managing Users

Adding a User

To add a user follow these steps…

5. Select Users in the left pane under the domain. Right-click and select New → User or click on the Add New User Icon on the toolbar.
6. Complete the appropriate information in the dialog and click Next.
7. Define a default password for the account and OK.
   (Note: the “User must change password on next login” checkbox will be set automatically.)
   (Note: the “Password never expires” may be set on admin accounts.)
8. Acknowledge the user account creation.

Some additional user definition not supported by the initial dialog needs to be completed.

9. Right-click on the user account and select Properties or click the Properties icon on the taskbar. This will open a detailed multi-tab dialog box to change any user setting.
10. On the General tab add a description consistent with previous descriptions (i.e. Admin Account, Saranam Staff, Instructors, Clients, …).
    (Change display name if desired. This defines the users identity shown on the Windows desktop.
11. Click on the MemberOf tab and add the user to the appropriate groups.
    (Note: the group must be pre-defined.
    (Note: to list available groups, click **Advanced**, then click **Find now**.)
12. Edit any other information as desired.

Terminating Users

When terminating a user, best practice involves simply **disabling** the account rather than deleting the account. This keeps the account in tact but prevents user login, which allows recovery in the future if necessary, as well as later administrator access if necessary.

Appendix W: Some Remote Synchronization (rsync) Notes

The Linux ***rsync*** command synchronizes files and folders, both locally and across remote systems. It supports a plethora of options making it extremely powerful, but beyond the scope of this document in terms of description. See ***man rsync*** for more information on the control and capabilities of ***rsync***.

Backup and Sync Script (/usr/local/bin/bsync)

The ***bsync*** script relies heavily on the use of ***rsync*** to perform the actual backup and cloning operations, as do the ***/usr/local/bin/backup*** and ***/usr/local/bin/backup_rpi*** scripts. These scripts simply wrap the rsync commands with logging and contextual info.

Automating Remote System Access

rsync has a nice feature of being able to sync across remote machines. This requires one establish a remote login. The following describes how to exchange public keys to enable automatic login and therefore automation of the synchronization. This description identifies the ***local machine*** (***black***) as the one executing ***rsync*** and the ***remote machine*** (***red***) as the one being access by ***rsync***. It assumes user ***pi*** on both machines, although you may configure any user in the same manner. It assumes too that you have ***SSH*** installed on both machines with a ***.ssh*** folder in each user area.

On the remote system you may have to establish a login shell for the user, if it doesn't exist, as well as make an SSH directory, and protect it.

```
su - <username>
usermod -s /bin/bash <username>
mkdir .ssh
chmod 700 .ssh
```

> ***Note: These are all optional commands. Normal users have a login shell and .ssh folder, but special accounts such as a designated backup operator may not.***

On the ***local machine***, from the user's (pi) home directory generate a key pair using:

```
ssh-keygen –t rsa
```

Just accept the defaults by pressing ***<Enter>*** at all of the prompts. This will create ***/home/pi/.ssh/id_rsa*** and ***/home/pi/.ssh/id_rsa.pub*** files. Then copy the public key file to the remote machine by

```
scp .ssh/id_rsa.pub pi@red:~/.ssh/authorized_keys
```

Test the ***remote system*** access from the local machine using an SSH login.

```
ssh pi@red
```

You should automatically connect and receive a welcome message. Exit the SSH login. Test rsync with a "dry-run" command that reports what it would do, but doesn't actually do anything.

```
rsync –avzr --dry-run test red:~/test .
```

This assumes a ***test*** folder on the local machine with some example files. It will sync files from ***/home/pi/test*** on ***black*** to ***/home/pi/test*** on ***red***.

You can now automate rsync (or the bsync script, for example) from a cronjob just like any other process.