

Meerkat system training



Linux administration

Jyri Soppela

24/09/2017

1. Introduction

- Main aim of the Meerkat system is to provide fast public health surveillance
- 2 days of training:
 - Day 1:
 - Server administration
 - Shell scripting
 - Managing server software
 - Networking servers and network security
 - Server security
 - Day 2:
 - Making changes to ODK forms
 - Using Meerkat APIs to retrieve data
 - Data analysis with Python and spreadsheet calculators

Contents

- 1. Introduction
- 2. Linux server administration
 - 2.1 Basic command line tools
 - 2.2 Monitoring server resources
 - 2.3 Installing software
- 3. Advanced command line techniques
 - 3.1 Input and output
 - 3.2 Common programs
- 4. Scripting
 - 4.1 When to script
 - 4.2 Script syntax

Exercise 1 a

As a first exercise, let's set up the computers to be able to access the training servers

- Download the PuTTY binary for your computer:

<http://www.putty.org/>

- Get the identity file `training.ppk` for the training servers.
- Use the following guide to get logged in:

<http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html>

2. Linux server administration

- 66% of public web sites are on servers that run Linux*
- Linux comes in several distributions that are curated by different institutions
- Linux is open source as are most of its distributions
- Meerkat uses Ubuntu distribution



```
jyri@isok1rahvi: ~  
jyri@isok1rahvi:~$ df -h  
Filesystem      Size  Used Avail Use% Mounted on  
udev            7.8G   0 7.8G   0% /dev  
tmpfs           1.6G  9.7M  1.6G   1% /run  
/dev/sdb5       443G   67G  354G  16% /  
tmpfs           7.9G  126M  7.7G   2% /dev/shm  
tmpfs           5.0M   4.0K  5.0M   1% /run/lock  
tmpfs           7.9G   0 7.9G   0% /sys/fs/cgroup  
tmpfs           1.6G   56K  1.6G   1% /run/user/1000  
/home/jyri/.Private 443G   67G  354G  16% /home/jyri  
jyri@isok1rahvi:~$ free -m  
              total        used        free      shared  buff/cache   available  
Mem:           15977         3060         9234         167        3682       12303  
Swap:          16311           0        16311  
jyri@isok1rahvi:~$
```

- Linux servers are used mostly via command line shell
- In case of Ubuntu, the operating system is the same on a server and a PC
- Command line tools have slight differences between distros, this training focuses on Ubuntu

*https://w3techs.com/technologies/overview/operating_system/all

2.1 Basic command line tools

```
jyri@isok1rahvi:~/Documents$
```

User

Host

Path

- When using Linux in shell, even the simplest things will be done by commands:
 - `ls` – List contents of current path
 - `cd` – Change current directory: `..` for parent directory
 - `cat` – Print file contents to standard output (command line)
 - `man` – Display manual page of command or program
 - Help pages are also often available by `<command> --help`
 - `find` – Find files
 - `grep` – Find row in file
 - `ssh` – Log in to another Linux machine
 - `apt` – Ubuntu package manager for installing software

Exercise 2 a

- Log in to your EC2 virtual server using Putty
- Use Git to download the training material from GitHub:

```
git clone https://github.com/meerkat-code/training.git
```

- Git is a version control software and a topic of another training, here we use it only to get training material
- Change current directory to

```
training/linux_administration/exercises/demo_locations
```

The .csv files in this folder include location data for country Demo. A clinic is in a district, which is in a region, which is in a zone.

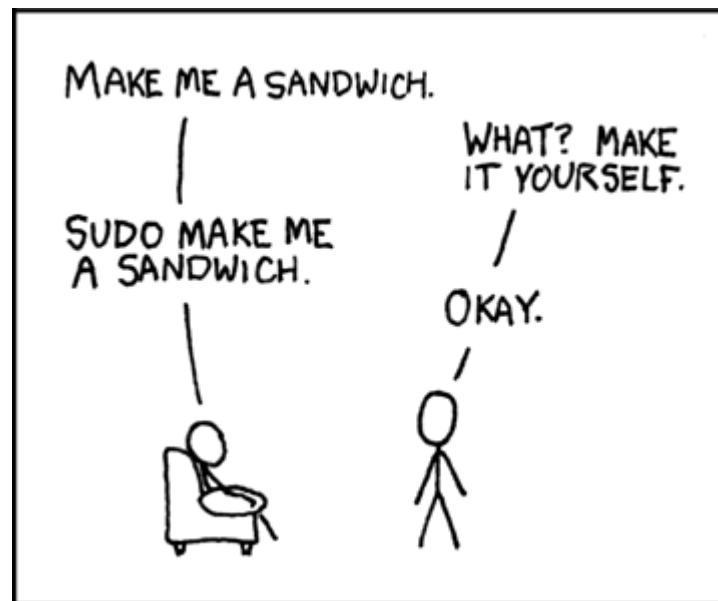
- Print the contents of file `demo_districts.csv` to standard output
- Print all rows in file `demo_clinics.csv` which include the word `Region 1`

2.2 Monitoring server resources

- Monitoring the usage of RAM, disk space, CPU or networking resources are very common tasks in server administration
- Widely available monitoring tools are:
- `df` – Show disk space usage and file systems
- `du` – Calculate the size of files and folders
- `free` – Show memory usage
- `top` – Show statistics on processes and CPU usage
- `netstat` – Show active Internet connections

2.3 User levels and installing software

- Normally you are allowed to perform operations which your user has rights to
 - This includes performing operations on files you have permissions to but not e.g. installing software
- If your user account is an administrator, you can run commands with an elevated access level by preceding a command with `sudo`



<https://xkcd.com/149/>

Exercise 2 b

- Find out the following about your virtual server:
 - Total amount of RAM installed in server
 - Total amount of disk space available for server
- Try installing the network monitoring tool `bmon` by running

```
apt install bmon
```
- Install `bmon` by using a command with elevated access
- You can try out the monitoring program with the command `bmon`

3.1 Input and output

Piping

Piping is a technique for forwarding the output of one command to the input of another command. Piping is done with the character `|`

Example:

```
cat demo_clinics.csv | grep "Clinic 1"
```

Redirecting

Redirecting is a technique for forwarding the output of one command into a file. Redirecting is done with the character `>`

Redirecting works the other way around too to use a file as command input. This is done with the character `<`

Examples:

```
cat demo_clinics.csv | grep "Clinic 1" > clinic_1.txt
```

```
grep "Clinic 1" < demo_clinics.csv
```

Exercise 2 b

- Find out the following about your virtual server:
 - Total amount of RAM installed in server
 - Total amount of disk space available for server
- Try installing the network monitoring tool `bmon` by running

```
apt install bmon
```
- Install `bmon` by using a command with elevated access
- You can try out the monitoring program with the command `bmon`

3.1 Input and output

Parameters

It is also possible to pass outputs of commands as parameters to other command. This is done with syntax `<command 1> $(<command 2>)`

Examples:

```
cat $(ls)
```

```
grep "Clinic 1" $(ls)
```

Wild card characters

On command line, you can replace characters with wildcard characters. A full stop `.` will denote any single character and a star `*` will denote any number of any characters. This is done with syntax `<command 1> $(<command 2>)`

Example:

```
grep "Clinic 1" $(find *.csv)
```

Exercise 3 a

- Go to directory

`training/linux_administration/exercises/demo_locations`

- Using the shell methods described above, perform the following tasks:
 - Print into a new file `demo_district_1_clinics.csv` all rows from file `demo_clinics.csv` that have word "District 1" on them
 - Get all the rows from file `demo_clinics.csv` that have the word "Clinic 2" on them. Forward these rows to program `sed` and use `sed` to replace word "Clinic 2" with word "Hospital 1". Write the output into a new file `demo_hospital_1.csv`

Exercise 3 b

- Go to directory

`training/linux_administration/exercises/demo_locations`

- Using AWK and methods described above, perform the following tasks:
 - Use AWK on `demo_clinics.csv` to print to output only the columns with clinic names and district names
 - File `demo_clinics.csv` has fields that have commas in them. These might become troublesome, so use AWK to print file `demo_clinics.csv` again as `demo_clinics.psv` using pipe character as a column separator instead of comma.

4 Scripting

- Shell commands can be written into a script file, which you can use to run several commands at the same time.

Anatomy of a shell script:

- First line defines the interpreter to use
- Program flow can be controlled by conditional clauses and loops
- Any command used in the command line can be used in a script file
- A running script file is considered a separate shell. An exit command will not exit the shell that started the script.

```
#!/bin/bash
for csv; do
    psv="${csv%.csv}.psv"
    echo converting "$csv" ...
    sed 's/,/|/g' "$csv" > "$psv"
done
echo all conversions successful
exit 0
```

4.1 When to script?

- Scripts can be a strong tool for automating tasks
- Scripting has some serious shortcomings, however:
 - Writing tests for scripts is not easy
 - The syntax is not as complete and water tight as with most other languages
 - Small errors may end up being very costly,
e.g. `rm -rf *` / instead of `rm -rf */`
(Do NOT try the above at home)
- Scripting is fast to implement and often the first step of automating a task
 - Once there's time to invest, scripts can be replaced by programs written in e.g. Python, Java or C

4.2 Script syntax

- The current user must have execute permissions on the script to be able to run it
- White spaces and line breaks are important syntax elements in shell scripting
- Some bash scripting syntax:

```
# hash is the comment character
```

```
var=1 # define a variable, note the lack of white spaces
```

```
echo ${var} # call a variable and in this case print its value
```

```
for var in val1 val2 valN; do # loop structure
```

```
    echo var
```

```
done
```

```
if [val1 == val2]; then # if statement
```

```
    echo True
```

```
fi
```

Bash reference manual: <https://www.gnu.org/software/bash/manual/bashref.html>

Exercise 3 b

- Go to directory

`training/linux_administration/exercises/demo_locations`

Use the command line tools, including scripts, to write a single flattened file that has the following columns for each clinic:

Device id, Clinic name, District name, Region name, Zone name