Introduction to Linux
for Cloud Computing

Gregor von Laszewski

Editor

laszewski@gmail.com

https://cloudmesh-community.github.io/pub/vonLaszewski-linux.epub

December 02, 2019 - 09:07 AM

Created by Cloudmesh & Cyberaide Bookmanager, https://github.com/cyberaide/bookmanager

Introduction to Linux

Gregor von Laszewski

INTRODUCTION TO LINUX

1 PREFACE

Mon Dec 2 09:07:16 EST 2019

1.1 DISCLAIMER

This book has been generated with Cyberaide Bookmanager.

Bookmanager is a tool to create a publication from a number of sources on the internet. It is especially useful to create customized books, lecture notes, or handouts. Content is best integrated in markdown format as it is very fast to produce the output.

Bookmanager has been developed based on our experience over the last 3 years with a more sophisticated approach. Bookmanager takes the lessons from this approach and distributes a tool that can easily be used by others.

The following shields provide some information about it. Feel free to click on them.

1.1.1 Acknowledgment

If you use bookmanager to produce a document you must include the following acknowledgement.

> *"This document was produced with Cyberaide Bookmanager developed by Gregor von Laszewski available at https://pypi.python.org/pypi/cyberaide-bookmanager. It is in the responsibility of the user to make sure an author acknowledgement section is included in your document. Copyright verification of content included in a book is responsibility of the book editor."*

The bibtex entry is

```
@Misc{www-cyberaide-bookmanager,
  author =   {Gregor von Laszewski},
```

```
  title =    {{Cyberaide Book Manager}},
  howpublished = {pypi},
  month =    apr,
  year =     2019,
  url={https://pypi.org/project/cyberaide-bookmanager/}
}
```

1.1.2 Extensions

We are happy to discuss with you bugs, issues and ideas for enhancements. Please use the convenient github issues at

- https://github.com/cyberaide/bookmanager/issues

Please do not file with us issues that relate to an editors book. They will provide you with their own mechanism on how to correct their content.

2 INTRODUCTION

2.1 LINUX SHELL

Learning Objectives

- Be able to know the basic commands to work in a Linux terminal.
- Get familiar with Linux Commands

In this chapter we introduce you to a number of useful shell commands. You may ask:

“Why is he so keen on telling me all about shells as I do have a beautiful GUI?”

You will soon learn that A GUI may not be that suitable if you like to manage 10, 100, 1000, 10000, … virtual machines. A commandline interface could be mcuh simpler and would allow scripting.

2.1.1 History

LINUX is a reimplementation by the community of UNIX which was developed in 1969 by Ken Thompson and Dennis Ritchie of Bell Laboratories and rewritten in C. An important part of UNIX is what is called the *kernel* which allows the software to talk to the hardware and utilize it.

In 1991 Linus Torvalds started developing a Linux Kernel that was initially targeted for PC’s. This made it possible to run it on Laptops and was later on further developed by making it a full Operating system replacement for UNIX.

2.1.2 Shell

One of the most important features for us will be to access the computer with the help of a *shell*. The shell is typically run in what is called a terminal and allows

interaction to the computer with commandline programs.

There are many good tutorials out there that explain why one needs a linux shell and not just a GUI. Randomly we picked the first one that came up with a google query. This is not an endorsement for the material we point to, but could be a worth while read for someone that has no experience in Shell programming:

http://linuxcommand.org/lc3_learning_the_shell.php

Certainly you are welcome to use other resources that may suite you best. We will however summarize in table form a number of useful commands that you may als find even as a RefCard.

http://www.cheat-sheets.org/#Linux

We provide in the next table a number of useful commands that you want to explore. For more information simply type man and the name of the command. If you find a useful command that is missing, please add it with a Git pull request.

.

Command	**Description**
man *command*	manual page for the *command*
apropos *text*	list all commands that have text in it
ls	Directory listing
ls -lisa	list details
tree	list the directories in graphical form
cd *dirname*	Change directory to *dirname*
mkdir *dirname*	create the directory
rmdir *dirname*	delete the directory
pwd	print working directory
rm *file*	remove the file
cp *a b*	copy file *a* to *b*
mv *a b*	move/rename file *a* to *b*

cat *a*	print content of file*a*
cat -n *filename*	print content of file*a* with
	line numbers
less *a*	print paged content of file *a*
head -5 *a*	Display first 5 lines of file *a*
tail -5 *a*	Display last 5 lines of file *a*
du -hs .	show in human readable form the space used by the current directory
df -h	show the details of the disk file system
wc *filename*	counts the word in a file
sort *filename*	sorts the file
uniq *filename*	displays only uniq entries in the file
tar -xvf *dir*	tars up a compressed version of the directory
rsync	faster, flexible replacement for rcp
gzip *filename*	compresses the file
gunzip *filename*	compresses the file
bzip2 *filename*	compresses the file with
	block-sorting
bunzip2 *filename*	uncompresses the file with block-sorting
clear	clears the terminal screen
touch *filename*	change file access and modification times or if file does not exist creates file
who	displays a list of users that are currently logged on, for each user the login name, date and time of login, tty name, and hostname if not local are displayed
	displays the users effective id see

whoami	also id
echo -n *string*	write specified arguments to standard output
date	displays or sets date & time, when invoked without arguments the current date and time are displayed
logout	exit a given session
exit	when issued at the shell prompt the shell will exit and terminate any running jobs within the shell
kill	terminate or signal a process by sending a signal to the specified process usually by the pid
ps	displays a header line followed by all processes that have controlling terminals
sleep	suspends execution for an interval of time specified in seconds
uptime	displays how long the system has been running
time *command*	times the command execution in seconds
find */ [-name] file-name.txt*	searches a specified path or directory with a given expression that tells the find utility what to find, if used as shown the find utility would search the entire drive for a file named file-name.txt
diff	compares files line by line
hostname	prints the name of the current host system
which	locates a program file in the users path
tail	displays the last part of the file

head	displays the first lines of a file
top	displays a sorted list of system processes
locate *filename*	finds the path of a file
grep *'word' filename*	finds all lines with the word in it
grep -v *'word' filename*	finds all lines without the word in it
chmod ug+rw *filename*	change file modes or Access Control Lists. In this example user and group are changed to read and write
chown	change file owner and group
history	a build-in command to list the past commands
sudo	execute a command as another user
su	substitute user identity
uname	print the operating system name
set -o emacs	tells the shell to use Emacs commands.
chmod go-rwx *file*	changes the permission of the file
chown *username file*	changes the ownership of the file
chgrp *group file*	changes the group of a file
fgrep *text filename*	searches the text in the given file
grep -R *text* .	recursively searches for xyz in all files
find . -name *.py	find all files with `.py` at the end
ps	list the running processes
kill -9 1234	kill the process with the id 1234
at	que commands for later execution
cron	daemon to execute scheduled commands
crontab	manage the time table for execution commands with cron

mount /dev/cdrom /mnt/cdrom	mount a filesystem from a cd rom to /mnt/cdrom
users	list the logged in users
who	display who is logged in
whoami	print the user id
dmesg	display the system message buffer
last	indicate last logins of users and ttys
uname	print operating system name
date	prints the current date and time
time *command*	prints the sys, real and user time
shutdown -h “shut down”	shutdown the computer
ping	ping a host
netstat	show network status
hostname	print name of current host system
traceroute	print the route packets take to network host
ifconfig	configure network interface parameters
host	DNS lookup utility
whois	Internet domain name and network number directory service
dig	DNS lookup utility
wget	non-interactive network downloader
curl	transfer a URL
ssh	remote login program
scp	remote file copy program
sftp	secure file transfer program
watch *command*	run any designated command at regular intervals
awk	program that you can use to select particular records in a file and

	perform operations on them
sed	stream editor used to perform basic text transformations
xargs	program that can be used to build and execute commands from STDIN
cat *some_file.json* | python -m json.tool	quick and easy JSON validator

2.1.3 The command man

On Linux you find a rich set of manual pages for thes commands. Try to pick one and execute:

```
$ man ls
```

You will see somthing like this

```
LS(1)                     BSD General Commands Manual                    LS(1)

NAME
     ls -- list directory contents

SYNOPSIS
     ls [-ABCFGHLOPRSTUW@abcdefghiklmnopqrstuwx1] [file ...]

DESCRIPTION

     For each operand that names a file of a type other than directory,
     ls displays its name as well as any requested, associated
     information.  For each operand that names a file of type directory,
     ls displays the names of files contained within that directory, as
     well as any requested, associated information.

     If no operands are given, the contents of the current directory are
     displayed.  If more than one operand is given, non-directory
     operands are displayed first; directory and non-directory operands
     are sorted separately and in lexicographical order.

     The following options are available:

     -@      Display extended attribute keys and sizes in long (-l) output.

     -1      (The numeric digit ``one''.)  Force output to be one entry
             per line.  This is the default when output is not to a terminal.

     -A      List all entries except for . and ...  Always set for the
             super-user.

     -a      Include directory entries whose names begin with a dot (.).


     ... on purpose cut ... instead try it yourslef
```

2.1.4 Multi-command execution

One of the important features is that one can execute multiple commands in the shell.

To execute command 2 once command 1 has finished use

```
command1; command2
```

To execute command 2 as soon as command 1 forwards output to stdout use

```
command1; command2
```

To execute command 1 in the background use

```
command1 &
```

2.1.5 Keyboard Shortcuts

These shortcuts will come in handy. Note that many overlap with emacs short cuts.

.

Keys	Description
Up Arrow	Show the previous command
Ctrl + z	Stops the current command
	Resume with `fg` in the foreground
	Resume with `bg` in the background
Ctrl + c	Halts the current command
Ctrl + l	Clear the screen
Ctrl + a	Return to the start of the line
Ctrl + e	Go to the end of the line
Ctrl + k	Cut everything after the cursor to a special clipboard
Ctrl + y	Paste from the special clipboard
Ctrl + d	Logout of current session, similar to exit

2.1.6 bashrc, bash_profile or zprofile

Usage of a particular command and all the attributes associated with it, use `man` command. Avoid using `rm -r` command to delete files recursively. A good way to avoid accidental deletion is to include the following in the file `.bash_profile` or `.zprofile` on macOS or `.bashrc` on other platforms:

```
alias rm='rm -i'
alias mv='mv -i'
alias h='history'
```

2.1.7 Makefile

Makefiles allow developers to coordinate the execution of code compilations. This not only includes C or C++ code, but any translation from source to a final format. For us this could include the creation of PDF files from latex sources, creation of docker images, and the creation of cloud services and their deployment through simple workflows represented in makefiles, or the coordination of execution targets.

As makefiles include a simple syntax allowing structural dependencies they can easily adapted to fulfill simple activities to be executed in repeated fashion by developers.

An example of how to use Makefiles for docker is provided at http://jmkhael.io/makefiles-for-your-dockerfiles/.

An example on how to use Makefiles for LaTeX is provided at https://github.com/cloudmesh/book/blob/master/Makefile.

Makefiles include a number of rules that are defined by a target name. Let us define a target called hello that prints out the string “Hello World”.

```
    hello:
        @echo "Hello World"
```

Important to remember is that the commands after a target are not indented just by spaces, but actually by a single TAB character. Editors such as emacs will be ideal to edit such Makefiles, while allowing syntax highlighting and easy manipulation of TABs. Naturally other editors will do that also. Please chose your editor of choice. One of the best features of targets is that they can depend on other targets. Thus, iw we define

```
    hallo: hello
```

```
	@echo "Hallo World"
```

our makefile will first execute hello and than all commands in hallo. As you can see this can be very useful for defining simple dependencies.

In addition we can define variables in a makefile such as

```
HELLO="Hello World"

hello:
    @echo $(HELLO)
```

and can use them in our text with $ invocations.

Moreover, in sophisticated Makefiles, we could even make the targets dependent on files and a target rules could be defined that only compiles those files that have changed since our last invocation of the Makefile, saving potentially a lot of time. However, for our work here we just use the most elementary makefiles.

For more information we recommend you to find out about it on the internet. A convenient reference card sis available at http://www.cs.jhu.edu/~joanne/unixRC.pdf.

2.1.8 chmod

The chmod command stand for *change mode* and changes the access permissions for a given file system object(s). It uses the following syntax: `chmod [options] mode[,mode] file1 [file2…]`. The option parameters modify how the process runs, including what information is outputted to the shell:

Option:	**Description:**
`-f`, `--silent`, `--quiet`	Forces process to continue even if errors occur
`-v`, `--verbose`	Outputs for every file that is processed
`-c`, `--changes`	Outputs when a file is changed
`--reference=RFile`	Uses RFile instead of Mode values
`-R`, `--recursive`	Make changes to objects in subdirectories as well
`--help`	Show help
`--version`	Show version information

Modes specify which rights to give to which users. Potential users include the user who owns the file, users in the file's Group, other users not in the file's Group, and all, and are abbreviated as `u`, `g`, `o`, and `a` respectively. More than one user can be specified in the same command, such as `chmod –v ug(operator)(permissions) file.txt`. If no user is specified, the command defaults to `a`. Next, a `+` or `-` indicates whether permissions should be added or removed for the selected user(s). The permissions are as follows:

Permission:	Description:
`r`	Read
`w`	Write
`x`	Execute file or access directory
`X`	Execute only if the object is a directory
`s`	Set the user or group ID when running
`t`	Restricted deletion flag or sticky mode
`u`	Specifies the permissions the user who owns the file has
`g`	Specifies the permissions of the group
`o`	Specifies the permissions of users not in the group

More than one permission can be also be used in the same command as follows:

```
$ chmod –v o+rw file.txt
```

Multiple files can also be specified:

```
$ chmod a-x,o+r file1.txt file2.txt
```

2.1.9 Exercises

E.Linux.1

> *Familiarize yourself with the commands*

E.Linux.2

> *Find more commands that you find useful and add them to this page.*

E.Linux.3

> *Use the sort command to sort all lines of a file while removing duplicates.*

E.Linux.4

> *Should there be other commands listed in the table with the Linux commands If so which? Create a pull request for them.*

E.Linux.5

> *Write a section explaining chmod. Use letters not numbers*

E.Linux.6

> *Write a section explaining chown. Use letters not numbers*

E.Linux.7

> *Write a section explaining su and sudo*

E.Linux.8

> *Write a section explaining cron, at, and crontab*

2.2 Perl One liners

Perl is a programming language that used to be very popular with system administrators. It predates Python. It has some very powerful regular expression abilities allowing you to easily do things on the commandline that woul otherwise thake many hours. Here ar some useful perl one line commands

Strip trailing whitespace from a file:

```
perl -lpe 's/\s*$//' FILENAME
```

Replace wrong quote

```
perl -i -p -e "s/’/'/g;"  *.md
```

Remove `^M` from file

```
perl -p -i -e 's/\r\n$/\n/g'
```

2.3 REFCARDS

Learning Objectives

- Obtain quickly information about technical aspects with the help of reference cards.

We present you with a list of useful short reference cards. This cards can be extremely useful to remind yourself about some important commands and features. Having them could simplify your interaction with the systems, We not only collected here some refcards about Linux, but also about other useful tools and services.

If you like to add new topics, let us know via your contribution (see the contribution section).

CheatSheets

- CheatSheets

Editors

- Emacs
- Vi
- Vim

Documentation

- LaTeX
- RST

Linux

- Linux
- Makefile
- Git

Cloud/Virtualization

- Openstack
- Openstack
- vagrant

SQL

- SQL

Languages

- R

Python

- Python
- PythonData
- Numpy/Pandas
- PythonTutorial
- Python
- Python
- PythonAPIIndex
- Python3

2.4 Secure Shell

Learning Objectives

- This is a very important sections of the book, studdy it carefully.
- learn how to use SSH keys
- Learn how to use ssh-add and ssh-keycahin so you only have to type in your password once

- Understand that each computer needs its own ssh key

Secure Shell is a network protocol allowing users to securely connect to remote resources over the internet. In many services we need to use SSH to assure that we protect he messages send between the communicating entities. Secure Shell is based on public key technology requiring to generate a public-private key pair on the computer. The public key will than be uploaded to the remote machine and when a connection is established during authentication the public private key pair is tested. If they match authentication is granted. As many users may have to share a computer it is possible to add a list of public keys so that a number of computers can connect to a server that hosts such a list. This mechanism builds the basis for networked computers.

In this section we will introduce you to some of the commands to utilize secure shell. We will reuse this technology in other sections to for example create a network of workstations to which we can log in from your laptop. For more information please also consult with the SSH Manual.

> ⚠ *Whatever others tell you, the private key should never be copied to another machine. You almost always want to have a passphrase protecting your key.*

2.4.1 ssh-keygen

The first thing you will need to do is to create a public private key pair. Before you do this check whether there are already keys on the computer you are using:

```
ls ~/.ssh
```

If there are files named id_rsa.pub or id_dsa.pub, then the keys are set up already, and we can skip the generating keys step. However you must know the passphrase of the key. If you forgot it you will need to recreate the key. However you will lose any ability to connect with the old key to the resources to which you uploaded the public key. So be careful.

To generate a key pair use the command ssh-keygen. This program is commonly available on most UNIX systems and most recently even Windows 10.

To generate the key, please type:

```
$ ssh-keygen -t rsa -C <comment>
```

The comment will remind you where the key has been created, you could for example use the hostname on which you created the key.

In the following text we will use *localname* to indicate the username on your computer on which you execute the command.

The command requires the interaction of the user. The first question is:

```
Enter file in which to save the key (/home/localname/.ssh/id_rsa):
```

We recommend using the default location ~/.ssh/ and the default name id_rsa. To do so, just press the enter key.

The second and third question is to protect your ssh key with a passphrase. This passphrase will protect your key because you need to type it when you want to use it. Thus, you can either type a passphrase or press enter to leave it without passphrase. To avoid security problems, you **MUST** chose a passphrase.

It will ask you for the location and name of the new key. It will also ask you for a passphrase, which you **MUST** provide. Please use a strong passphrase to protect it appropriately. Some may advise you (including teachers and TA's) to not use passphrases. This is **WRONG** as it allows someone that gains access to your computer to also gain access to all resources that have the public key. Only for some system related services you may create passwordless keys, but such systems need to be properly protected.

> ⚠ *Not using passphrases poses a security risk!*

Make sure to not just type return for an empty passphrase:

```
Enter passphrase (empty for no passphrase):
```

and:

```
Enter same passphrase again:
```

If executed correctly, you will see some output similar to:

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/localname/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/localname/.ssh/id_rsa.
Your public key has been saved in /home/localname/.ssh/id_rsa.pub.
The key fingerprint is:
34:87:67:ea:c2:49:ee:c2:81:d2:10:84:b1:3e:05:59 localname@indiana.edu

+--[ RSA 2048]----+
|.+...Eo= .       |
| ..=.o + o +o    |
|O.  = ......     |
| = .   . .       |
+-----------------+
```

Once, you have generated your key, you should have them in the `.ssh` directory. You can check it by:

```
$ cat ~/.ssh/id_rsa.pub
```

If everything is normal, you will see something like:

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQCXJH2iG2FMHqC6T/U7uB8kt
6KlRh4kUOjgw9sc4Uu+Uwe/kshuispauhfsjhfm,anf6787sjgdkjsgl+EwD0
thkoamyi0VvhTVZhj61pTdhyl1t8hlkoL19JVnVBPP5kIN3wVyNAJjYBrAUNW
4dXKXtmfkXp98T3OW4mxAtTH434MaT+QcPTcxims/hwsUeDAVKZY7UgZhEbiE
xxkejtnRBHTipi0W03W05TOUGRW7EuKf/4ftNVPilCO4DpfY44NFG1xPwHeim
Uk+t9h48pBQj16FrUCp0rS02Pj+4/9dNeS1kmNJu5ZYS8HVRhvuoTXuAY/UVc
ynEPUegkp+qYnR user@myemail.edu
```

The directory `~/.ssh` will also contain the private key `id_rsa` which you must not share or copy to another computer.

> ⚠ *Never, copy your private key to another machine or check it into a repository!*

To see what is in the .ssh directory, please use

```
$ ls ~/.ssh
```

Typically you will se a list of files such as

```
authorized_keys
id_rsa
id_rsa.pub
```

```
known_hosts
```

In case you need to change your change passphrase, you can simply run `ssh-keygen -p` command. Then specify the location of your current key, and input (old and) new passphrases. There is no need to re-generate keys:

```
ssh-keygen -p
```

You will see the following output once you have completed that step:

```
Enter file in which the key is (/home/localname/.ssh/id_rsa):
Enter old passphrase:
Key has comment '/home/localname/.ssh/id_rsa'
Enter new passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved with the new passphrase.
```

2.4.2 ssh-add

Often you wil find wrong information about passphrases on the internet and people recommending you not to use one. However it is in almost all cases better to create a key pair and use `ssh-add` to add the key to the current session so it can be used in behalf of you. This is accomplished with an agent.

The `ssh-add` command adds SSH private keys into the SSH authentication agent for implementing single sign-on with SSH. ssh-add allows the user to use any number of servers that are spread across any number of organizations, without having to type in a password every time when connecting between servers. This is commonly used by system administrators to login to multiple server.

`ssh-add` can be run without arguments. When run without arguments, it adds the following default files if they do exist:

- `~/.ssh/identity` - Contains the protocol version 1 RSA authentication identity of the user.
- `~/.ssh/id_rsa` - Contains the protocol version 1 RSA authentication identity of the user.
- `~/.ssh/id_dsa` - Contains the protocol version 2 DSA authentication identity of the user.
- `~/.ssh/id_ecdsa` - Contains the protocol version 2 ECDSA authentication identity of the user.

To add a key you can provide the path of the key file as an argument to ssh-add.

For example,

```
ssh-add ~/.ssh/id_rsa
```

would add the file `~/.ssh/id_rsa`

If the key being added has a passphrase, `ssh-add` will run the `ssh-askpass` program to obtain the passphrase from the user. If the `SSH_ASKPASS` environment variable is set, the program given by that environment variable is used instead.

Some people use the `SSH_ASKPASS` environment variable in scripts to provide a passphrase for a key. The passphrase might then be hard-coded into the script, or the script might fetch it from a password vault.

The command line options of `ssh-add` are as follows:

Option	Description
`-c`	Causes a confirmation to be requested from the user every time the added identities are used for authentication. The confirmation is requested using ssh-askpass.
`-D`	Deletes all identities from the agent.
`-d`	Deletes the given identities from the agent. The private key files for the identities to be deleted should be listed on the command line.
`-e` pkcs11	Remove key provided by pkcs11
`-L`	Lists public key parameters of all identities currently represented by the agent.
`-l`	Lists fingerprints of all identities currently represented by the agent.
`-s` pkcs11	Add key provided by pkcs11.
`-t` life	Sets the maximum time the agent will keep the given key. After the timeout expires, the key will be automatically removed from the agent. The default value is in seconds, but can be suffixed for m for minutes, h for hours, d for days, or w for weeks.
`-X`	Unlocks the agent. This asks for a password to unlock.

`-x`	Locks the agent. This asks for a password; the password is required for unlocking the agent. When the agent is locked, it cannot be used for authentication.

2.4.3 SSH Add and Agent

To not always type in your password, you can use `ssh-add` as previously discussed

It prompts the user for a private key passphrase and add it to a list of keys managed by the ssh-agent. Once it is in this list, you will not be asked for the passphrase as long as the agent is running.with your public key. To use the key across terminal shells you can start an ssh agent.

To start the agent please use the following command:

```
$ eval `ssh-agent`
```

or use

```
$ eval "$(ssh-agent -s)"
```

It is important that you use the backquote, located under the tilde (US keyboard), rather than the single quote. Once the agent is started it will print a PID that you can use to interact with later

To add the key use the command

```
$ ssh-add
```

To remove the agent use the command

```
kill $SSH_AGENT_PID
```

To execute the command upon logout, place it in your `.bash_logout` (assuming you use bash).

On OSX you can also add the key permanently to the keychain if you do toe following:

```
ssh-add -K ~/.ssh/id_rsa
```

Modify the file `.ssh/config` and add the following lines:

```
Host *
  UseKeychain yes
  AddKeysToAgent yes
  IdentityFile ~/.ssh/id_rsa
```

2.4.3.1 Using SSH on Mac OS X

Mac OS X comes with an ssh client. In order to use it you need to open the `Terminal.app` application. Go to `Finder`, then click `Go` in the menu bar at the top of the screen. Now click `Utilities` and then open the `Terminal` application.

2.4.3.2 Using SSH on Linux

All Linux versions come with ssh and can be used right from the terminal.

2.4.3.3 Using SSH on Raspberry Pi 3/4

SSH is available on Raspbian. However, to ssh into the PI you have to activate it via the configuration menu.

2.4.3.4 Accessing a Remote Machine

Once the key pair is generated, you can use it to access a remote machine. To dod so the public key needs to be added to the `authorized_keys` file on the remote machine.

The easiest way to do tis is to use the command `ssh-copy-id`.

```
$ ssh-copy-id user@host
```

Note that the first time you will have to authenticate with your password.

Alternatively, if the ssh-copy-id is not available on your system, you can copy the file manually over SSH:

```
$ cat ~/.ssh/id_rsa.pub | ssh user@host 'cat >> .ssh/authorized_keys'
```

Now try:

```
$ ssh user@host
```

and you will not be prompted for a password. However, if you set a passphrase when creating your SSH key, you will be asked to enter the passphrase at that time (and whenever else you log in in the future). To avoid typing in the password all the time we use the ssh-add command that we described earlier.

```
$ ssh-add
```

2.4.4 SSH Port Forwarding 🅾

🅾 TODO: Add images to illustrate the concepts

SSH Port forwarding (SSH tunneling) creates an encrypted secure connection between a local computer and a remote computer through which services can be relayed. Because the connection is encrypted, SSH tunneling is useful for transmitting information that uses an unencrypted protocol.

2.4.4.1 Prerequisites

- Before you begin, you need to check if forwarding is allowed on the SSH server you will connect to.
- You also need to have a SSH client on the computer you are working on.

If you are using the OpenSSH server:

```
$ vi /etc/ssh/sshd_config
```

and look and change the following:

```
AllowTcpForwarding = Yes
GatewayPorts = Yes
```

Set the `GatewaysPorts` variable only if you are going to use remote port forwarding (discussed later in this tutorial). Then, you need to restart the server for the change to take effect.

2.4.4.2 How to Restart the Server

If you are on:

- Linux, depending upon the init system used by your distribution, run:

```
$ sudo systemctl restart sshd
$ sudo service sshd restart
```

Note that depending on your distribution, you may have to change the service to ssh instead of sshd.

- Mac, you can restart the server using:

```
$ sudo launchctl unload /System/Library/LaunchDaemons/ssh.plist
$ sudo launchctl load -w /System/Library/LaunchDaemons/ssh.plist
```

- Windows and want to set up a SSH server, have a look at MSYS2 or Cygwin.

2.4.4.3 Types of Port Forwarding

There are three types of SSH Port forwarding:

2.4.4.4 Local Port Forwarding

Local port forwarding lets you connect from your local computer to another server. It allows you to forward traffic on a port of your local computer to the SSH server, which is forwarded to a destination server. To use local port forwarding, you need to know your destination server, and two port numbers.

Example 1:

```
$ ssh -L 8080:www.cloudcomputing.org:80 <host>
```

Where `<host>` should be replaced by the name of your laptop. The -L option specifies local port forwarding. For the duration of the SSH session, pointing your browser at `http://localhost:8080/` would send you to `http://cloudcomputing.com`

Example 2:

This example opens a connection to the www.cloudcomputing.com jump server, and forwards any connection to port 80 on the local machine to port 80 on `intra.example.com`.

```
$ ssh -L 80:intra.example.com:80 www.cloudcomputing.com
```

Example 3:

By default, anyone (even on different machines) can connect to the specified port on the SSH client machine. However, this can be restricted to programs on the same host by supplying a bind address:

```
$ ssh -L 127.0.0.1:80:intra.example.com:80 www.cloudcomputing.com
```

Example 4:

```
$ ssh -L 8080:www.Cloudcomputing.com:80 -L 12345:cloud.com:80 <host>
```

This would forward two connections, one to `www.cloudcomputing.com`, the other to `www.cloud.com`. Pointing your browser at `http://localhost:8080/` would download pages from www.cloudcomputing.com, and pointing your browser to `http://localhost:12345/` would download pages from www.cloud.com.

Example 5:

The destination server can even be the same as the SSH server.

```
$ ssh -L 5900:localhost:5900 <host>
```

The LocalForward option in the OpenSSH client configuration file can be used to configure forwarding without having to specify it on command line.

2.4.4.5 Remote Port Forwarding

Remote port forwarding is the exact opposite of local port forwarding. It forwards traffic coming to a port on your server to your local computer, and then it is sent to a destination. The first argument should be the remote port where traffic will be directed on the remote system. The second argument should be the address and port to point the traffic to when it arrives on the local system.

```
$ ssh -R 9000:localhost:3000 user@clodcomputing.com
```

SSH does not by default allow remote hosts to forwarded ports. To enable remote forwarding add the following to: `/etc/ssh/sshd_config`

```
GatewayPorts yes
```

```
$ sudo vim /etc/ssh/sshd_config
```

and restart SSH

```
$ sudo service ssh restart
```

After completing the previous steps you should be able to connect to the server remotely, even from your local machine. `ssh -R` first creates an SSH tunnel that forwards traffic from the server on port 9000 to your local machine on port 3000.

2.4.4.6 Dynamic Port Forwarding

Dynamic port forwarding turns your SSH client into a SOCKS proxy server. SOCKS is a little-known but widely-implemented protocol for programs to request any Internet connection through a proxy server. Each program that uses the proxy server needs to be configured specifically, and reconfigured when you stop using the proxy server.

```
$ ssh -D 5000 user@clodcomputing.com
```

The SSH client creates a SOCKS proxy at port 5000 on your local computer. Any traffic sent to this port is sent to its destination through the SSH server.

Next, you'll need to configure your applications to use this server. The *Settings* section of most web browsers allow you to use a SOCKS proxy.

2.4.4.7 ssh config

Defaults and other configurations can be added to a configuration file that is placed in the system. The ssh program on a host receives its configuration from

- the command line options
- a user-specific configuration file: `~/.ssh/config`
- a system-wide configuration file: `/etc/ssh/ssh_config`

Next we provide an example on how to use a config file

2.4.4.8 Tips

Use SSH keys

- You will need to use ssh keys to access remote machines

No blank passphrases

- In most cases you must use a passphrase with your key. In fact if we find that you use passwordless keys to futuresystems and to chameleon cloud resources, we may elect to give you an*F* for the assignment in question. There are some exceptions, but they will be clearly communicated to you in class. You will as part of your cloud drivers license test explain how you gain access to futuresystems and chameleon to explicitly explain this point and provide us with reasons what you can not do.

A key for each server

- Under no circumstances copy the same private key on multiple servers. This violates security best practices. Create for each server a new private key and use their public keys to gain access to the appropriate server.

Use SSH agent

- So as to not to type in all the time the passphrase for a key, we recommend using ssh-agent to manage the login. This will be part of your cloud drivers license.

  But shut down the ssh-agent if not in use

keep an offline backup, put encrypt the drive

- You may for some of our projects need to make backups of private keys on other servers you set up. If you like to make a backup you can do so on a USB stick, but make sure that access to the stick is encrypted. Do not store anything else on that key and look it in a safe place. If you lose the stick, recreate all keys on all machines.

2.4.4.9 References

- The Secure Shell: The Definitive Guide, 2 Ed (O'Reilly and Associates)

2.5 SSH AND BASH ON WINDOWS

For this class we recommend that you use a virtual machine via virtual box and use the Linux ssh instructions. The information here is just provided for completeness and no support will be offered for native windows support.

Windows users need to have some special software to be able to use the SSH commands. If you have one that you are comfortable with and know how to setup key pairs and access the contents of your public key, please feel free to use it.

On Windows you have a couple of options on running Linux commands such as ssh. At this time it may be worth while to try the OpenSSH Client available for Windows, although it is in beta. If you like to use other methods we have included alternatives.

2.5.1 OpenSSH Client on Windows

software to be able to run it directly from the Windows commandline including PowerShell.

However it is as far as we know not activated by default so you need to follow some setup scripts. Also this software is considered beta and its development and issues can be found at

- https://github.com/PowerShell/openssh-portable

Fortunately, the software is already distributed with Winodws 10, but may not yet been activated. What you have to do is to install it by going to `Settings > Apps` and click `Manage optional features` under `Apps & features`.

Next, Click on the `Add feature`. You will be presented with a list in which you scroll down, till you find `OpenSSH Client (Beta)`. Click on it and invoke `Install`.

After the install has completed, you can use the `ssh` command. Just type it in the commandshell or PowerShell

```
PS C:\Users\gregor> ssh
```

Naturally you can now use it just as on Linux or macOS. and use it to login to other resources

```
PS C:\Users\gregor> ssh myname@computer.example.com
```

see also the MS SSH Guide for the newest up dates.

Due to the availability of SSH on Windows 10, we no longer recommend using Cygwin SSH, PuTTY or Chocolatey. However we kept thise sections here for completness.

2.5.2 GitBash

A realy great tool for Windows is made avalable via

- https://gitforwindows.org/

Here you can find gitbash that provides you with a terminal in which you can natively execute linux commands such as `cd`, `ls` and many more. It also includes `ssh` and `ssh-keygen`. which you will need if you want o interface with Linux machines hosted in a cloud.

You can also enable the Git GUI as you may be used to doing things form GUI's. However soon you will find out why in this class we typicaly do not much via GUIs. However if you like them you can also integrate git in the Windows Explorer. This could be beneficial fo you during development of your project or keep up with what others do on git.

2.5.2.1 Makefiles on Windows

Makefiles can easily be accessed also on windows while installing gitbash. Please reed to the internet or search in this handbook for more information about gitbash.

🅾 Please contribute to this section on how to install make on wondows natively. Here is some information to start with Make on Windows

2.5.3 Using SSH from Cygwin

One established way of using ssh is from using cygwin.

http://cygwin.com/install.html Cygwin contains a collection of GNU and Open Source tools providing Linux like functionality on Windows. A DLL is available that exposes the POSIX API functionality.

A list of supported commands is available at

https://cygwin.com/packages/package_list.html Please be minded that in order for cygwin to function easily the Windows user name should not include spaces. However, as the setup in windows encourages to use the full name when you buy and setup a machine it may not be convenient to use. However, we just recommend that you create yourself a new username and use this if you like to use cygwin.

You can selectively install from the cygwin setup terminal which software you like to use, obviously you may want to use ssh

2.5.4 SSH from putty

As you will see the process is somewhat cumbersome and when you compare it with the commandline tools available, we do recommend using them instead.

PuTTY allows you to access the SSH, Telnet and Rlogin network protocols from windows.

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html Although PuTTY has been out there for many years and served the community well, it is not following the standard ssh command line syntax when invoked from a command shell.

```
putty -ssh user@host.name
```

In addition to using ssh, it also provides a copy command.

```
pscp user@host.name:"\"remote filename with spaces\"" local_filename
```

Putty is best known for its GUI configuration application to manage several machines as demonstrated next. Once you have downloaded it and opened PuTTYgen, you will be presented with a a key generator window (images provided by chameleon cloud) (see Figure [F:putty-key]).

To generate a key you click the *Generate* button which is blue. The PuTTY Key Generator (see Figure [F:putty-pass]) will then ask you to move your mouse around the program's blank space to generate "randomness" for your key. You must enter a "Key passphrase" and then confirm the passphrase.

Next you need to save both the public and private keys into a file of your choice using the "Save public key" and "Save private key" buttons. We suggest you name something obvious like "public_key.pub" and "private_key" so that you can distinguish between the two.

Before closing this window, select the entire public key and copy it with "Control-C". Please note that everything should be copied, including "ssh-rsa". This will be used when importing the key pair to Openstack.

At this time, the public key has been created and copied. Now you can use the public key and upload it to systems you like to login to.

2.5.5 Chocolatey

Another approach is to use it in Powershell with the help of chocolatey. Other options may be better suited for you and we leave it up to you to make this decision.

Chocolatey is a software management tool that mimics the install experience that you have on Linux and macOS. It has a repository with many packages. The packages are maintained by the community and you need to evaluate security implications when installing packages hosted on chocolatey just as you have to do if you install software on Linux and macOS from their repositories. Please be aware that there could be malicious code offered in the chocolatey repository although the distributors try to remove them.

The installation is sufficiently explained at

https://chocolatey.org/install Once installed you have a command choco and you should make sure you have the newest version with:

```
choco upgrade chocolatey
```

Now you can browse packages at

https://chocolatey.org/packages Search for openssh and see the results. You may find different versions. Select the one that most suits you and satisfies your security requirements as well as your architecture. Lets assume you chose the Microsoft port, than you can install it with:

```
choco install openssh
```

Naturally, you can also install cygwin and ptty over chocolatey. A list of packages can be found at

https://chocolatey.org/packages Packages of interest include

- emacs: choco install emacs
- pandoc: choco install pandoc
- LaTeX: choco install miktex
- jabref: choco install jabref
- pycharm: choco install pycharm-community
- lyx: choco install lyx
- python 2: choco install python2
- python 3: choco install python
- pip: choco install pip
- virtualbox: choco install virtualbox
- vagrant: choco install vagrant

Before installing any of them evaluate if you need them and identify security risks.

2.6 SSH and Bash on Windows

For this class we recommend that you use a virtual machine via virtual box and use the Linux ssh instructions. The information here is just provided for completeness and no support will be offered for native windows support.

Windows users need to have some special software to be able to use the SSH commands. If you have one that you are comfortable with and know how to setup key pairs and access the contents of your public key, please feel free to use it.

On Windows you have a couple of options on running Linux commands such as ssh. At this time it may be worth while to try the OpenSSH Client available for Windows, although it is in beta. If you like to use other methods we have included alternatives.

2.6.1 OpenSSH Client on Windows

software to be able to run it directly from the Windows commandline including PowerShell.

However it is as far as we know not activated by default so you need to follow some setup scripts. Also this software is considered beta and its development and issues can be found at

- https://github.com/PowerShell/openssh-portable

Fortunately, the software is already distributed with Winodws 10, but may not yet been activated. What you have to do is to install it by going to `Settings > Apps` and click `Manage optional features` under `Apps & features`.

Next, Click on the `Add feature`. You will be presented with a list in which you scroll down, till you find `OpenSSH Client (Beta)`. Click on it and invoke `Install`.

After the install has completed, you can use the `ssh` command. Just type it in the commandshell or PowerShell

```
PS C:\Users\gregor> ssh
```

Naturally you can now use it just as on Linux or macOS. and use it to login to other resources

```
PS C:\Users\gregor> ssh myname@computer.example.com
```

see also the MS SSH Guide for the newest up dates.

Due to the availability of SSH on Windows 10, we no longer recommend using Cygwin SSH, PuTTY or Chocolatey. However we kept thise sections here for completness.

2.6.2 GitBash

A realy great tool for Windows is made avalable via

- https://gitforwindows.org/

Here you can find gitbash that provides you with a terminal in which you can natively execute linux commands such as `cd`, `ls` and many more. It also includes `ssh` and `ssh-keygen`. which you will need if you want o interface with Linux machines hosted in a cloud.

You can also enable the Git GUI as you may be used to doing things form GUI's. However soon you will find out why in this class we typicaly do not much via GUIs. However if you like them you can also integrate git in the Windows Explorer. This could be beneficial fo you during development of your project or keep up with what others do on git.

2.6.2.1 Makefiles on Windows

Makefiles can easily be accessed also on windows while installing gitbash. Please reed to the internet or search in this handbook for more information about gitbash.

🅾 Please contribute to this section on how to install make on wondows natively. Here is some information to start with Make on Windows

2.6.3 Using SSH from Cygwin

One established way of using ssh is from using cygwin.

http://cygwin.com/install.html Cygwin contains a collection of GNU and Open Source tools providing Linux like functionality on Windows. A DLL is available that exposes the POSIX API functionality.

A list of supported commands is available at

https://cygwin.com/packages/package_list.html Please be minded that in order for cygwin to function easily the Windows user name should not include spaces. However, as the setup in windows encourages to use the full name when you buy and setup a machine it may not be convenient to use. However, we just

recommend that you create yourself a new username and use this if you like to use cygwin.

You can selectively install from the cygwin setup terminal which software you like to use, obviously you may want to use ssh

2.6.4 SSH from putty

As you will see the process is somewhat cumbersome and when you compare it with the commandline tools available, we do recommend using them instead.

PuTTY allows you to access the SSH, Telnet and Rlogin network protocols from windows.

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html Although PuTTY has been out there for many years and served the community well, it is not following the standard ssh command line syntax when invoked from a command shell.

```
putty -ssh user@host.name
```

In addition to using ssh, it also provides a copy command.

```
pscp user@host.name:"\"remote filename with spaces\"" local_filename
```

Putty is best known for its GUI configuration application to manage several machines as demonstrated next. Once you have downloaded it and opened PuTTYgen, you will be presented with a a key generator window (images provided by chameleon cloud) (see Figure [F:putty-key]).

To generate a key you click the *Generate* button which is blue. The PuTTY Key Generator (see Figure [F:putty-pass]) will then ask you to move your mouse around the program's blank space to generate "randomness" for your key. You must enter a "Key passphrase" and then confirm the passphrase.

Next you need to save both the public and private keys into a file of your choice using the "Save public key" and "Save private key" buttons. We suggest you name something obvious like "public_key.pub" and "private_key" so that you can distinguish between the two.

Before closing this window, select the entire public key and copy it with “Control-C”. Please note that everything should be copied, including “ssh-rsa”. This will be used when importing the key pair to Openstack.

At this time, the public key has been created and copied. Now you can use the public key and upload it to systems you like to login to.

2.6.5 Chocolatey

Another approach is to use it in Powershell with the help of chocolatey. Other options may be better suited for you and we leave it up to you to make this decision.

Chocolatey is a software management tool that mimics the install experience that you have on Linux and macOS. It has a repository with many packages. The packages are maintained by the community and you need to evaluate security implications when installing packages hosted on chocolatey just as you have to do if you install software on Linux and macOS from their repositories. Please be aware that there could be malicious code offered in the chocolatey repository although the distributors try to remove them.

The installation is sufficiently explained at

https://chocolatey.org/install Once installed you have a command choco and you should make sure you have the newest version with:

```
choco upgrade chocolatey
```

Now you can browse packages at

https://chocolatey.org/packages Search for openssh and see the results. You may find different versions. Select the one that most suits you and satisfies your security requirements as well as your architecture. Lets assume you chose the Microsoft port, than you can install it with:

```
choco install openssh
```

Naturally, you can also install cygwin and ptty over chocolatey. A list of packages can be found at

https://chocolatey.org/packages Packages of interest include

- emacs: choco install emacs
- pandoc: choco install pandoc
- LaTeX: choco install miktex
- jabref: choco install jabref
- pycharm: choco install pycharm-community
- lyx: choco install lyx
- python 2: choco install python2
- python 3: choco install python
- pip: choco install pip
- virtualbox: choco install virtualbox
- vagrant: choco install vagrant

Before installing any of them evaluate if you need them and identify security risks.

2.7 ZSH

Z shell (zsh) is an alternative to bash. It is used as an interactive shell or command interpreter. Zsh has been chosen by apple as a replacement for bash. A large number of plugins for zsh is avalable at the Web site Oh My Zsh.

Features of zsh include:

- commandline completion
- global history that can be shared in shells
- build in file globing
- multiline commands
- spell correction
- compatibillity modes to impersonate other shells
- themes for prompts
- in addition to which a wher ecommand
- shortcut to names directories with ~

In principal it does not matter much whcih shell you use as long as you use to set up your envireonment properly. While bash zupporst .bash_profile and .bashrc,

zsh supports ~/.zprofile and ~/.zshrc

A good overview of the loading process is documented at

- https://medium.com/@rajsek/zsh-bash-startup-files-loading-order-bashrc-zshrc-etc-e30045652f2e

Setting up zsh on an older OSX is relatively simple.

```
$ brew install zsh
```

To add Oh My Zsh you can do:

```
$ sh -c "$(curl -fsSL https://raw.githubusercontent.com/robbyrussell/oh-my-zsh/master/tools/install.sh)"
```

To chnge the default shell to zsh you can than execute

```
$ chsh -s $(which zsh)
```

To activate the shell you can as usal do

```
$ source ~/.zshrc
```

However if you start a new terminal, you do not have to do this as it is added automatically.

To use a number of useful plugins, you can activate them with

```
$ plugins=(osx git colored-man colorize pip python brew zsh-syntax-highlighting zsh-autosuggestions)
```

If you like to change the theme you can find a large number at

- https://github.com/ohmyzsh/ohmyzsh/wiki/Themes

2.7.1 Other operating systems

For other operationg systems see

- https://github.com/ohmyzsh/ohmyzsh/wiki/Installing-ZSH

2.7.2 zsh on Windows 10

To install zsh on Windows 10, please look at

- https://www.howtogeek.com/258518/how-to-use-zsh-or-another-shell-in-windows-10/

2.7.3 Exercises

E.SSH.1:

> *Create an SSH key pair*

E.SSH.2:

> *Upload the public key to git repository you use.*

E.SSH.3:

> *What is the output of a key that has a passphrase when executing the following command. Test it out on your key*

```
$ grep ENCRYPTED ~/.ssh/id_rsa
```

E.SSH.4

> *Get an account on futuresystems.org (if you are authorized to do so). Upload your key to https://futuresystems.org. Login to india.futuresystems.org. Note that this could take some time as administrators need to approve you. Be patient.*

E.SSH.5:

> *What can happen if you copy your private key to a machine on the network?*

E.SSH.6:

> *Should I share my provate key with others?*

E.SSH.7:

> *Assume I participate in a video conference call and I accidently share my private key. What should I do?*

E.SSH.8:

> *Assume I participate in a video conference call and I accidently share my public key. What should I do?*

3 REFERENCES