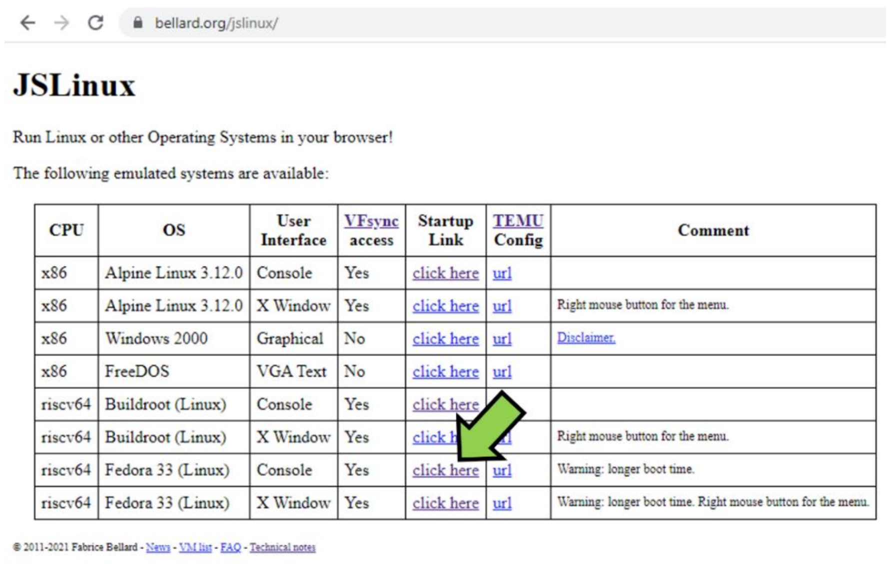
Session 1: Introduction to Unix shells

IN1011 tutorial exercises give examples of the operating system concepts we learn in the lectures, using the Linux operating system. Today's session will introduce you to Unix *command-line interfaces* – in particular, *shells* – and how to use Linux by issuing commands in shells.

Task 0: accessing Linux from your computer

The exercises will use one or both of the following two Linux environments:

1. ***JSLinux* (https://bellard.org/jslinux/):** online emulations of Linux variants, made available by Fabrice Bellard and written in *Javascript*. There are a couple of emulated operating systems available which you are encouraged to explore. However, for the IN1011 tutorials, click on the link for the Fedora33 console (see Figure 1). This will start a Linux console embedded in a webpage. For more information about JSLinux, see https://bellard.org/jslinux/tech.html . Caution: we advise you to not use your personal information with JSLinux emulators, nor use JSLinux to access your personal data; doing so may be putting your personal data at risk;
2. **City University's *ssh*-enabled Linux server[1]:** you can connect securely to the server as follows. **1)** make sure you are on City's network; i.e., you are connected to the IT network on campus or you are connected via *virtual private network* (VPN)[2]; **2)** open the command terminal in Windows 10 or a terminal in OSX. Type `ssh cityusername@Linux.city.ac.uk` at the command prompt, inserting your City university login username for "*cityusername*", and press the Enter key; **3)** If this is your first time connecting to the Linux server from your computer, you will be asked to confirm that you want to proceed. Indicate "yes" and enter your City-login password. You should then be presented with a flashing prompt indicating that you have successfully logged into the *Bash* shell, and the server is ready to receive your Unix commands (see Figure 2).



bellard.org/jslinux/

JSLinux

Run Linux or other Operating Systems in your browser!

The following emulated systems are available:

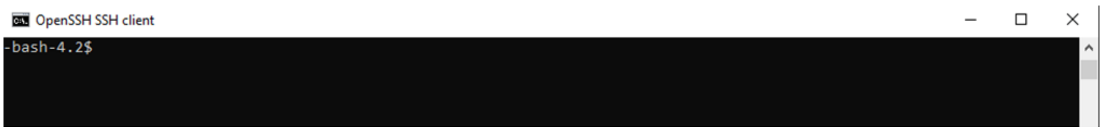
CPU	OS	User Interface	VFsync access	Startup Link	TEMU Config	Comment
x86	Alpine Linux 3.12.0	Console	Yes	click here	url	
x86	Alpine Linux 3.12.0	X Window	Yes	click here	url	Right mouse button for the menu.
x86	Windows 2000	Graphical	No	click here	url	Disclaimer.
x86	FreeDOS	VGA Text	No	click here	url	
riscv64	Buildroot (Linux)	Console	Yes	click here	url	
riscv64	Buildroot (Linux)	X Window	Yes	click here	url	Right mouse button for the menu.
riscv64	Fedora 33 (Linux)	Console	Yes	click here	url	Warning: longer boot time.
riscv64	Fedora 33 (Linux)	X Window	Yes	click here	url	Warning: longer boot time. Right mouse button for the menu.

© 2011-2021 Fabrice Bellard - News - VM list - FAQ - Technical notes

Figure 1: JSLinux hyperlink to Fedora33 shell

[1] The *secure shell protocol* (ssh) defines encrypted communication between a client and server. In particular, it allows a client to securely run shell commands on a remote server. The following video from the Youtube channel Computerphile gives a quick, detailed intro to ssh: https://www.youtube.com/watch?v=ORcvSkgdA58

[2] For instructions on how to connect to City's network from a Windows computer or Mac using a VPN, see https://cityuni.service-now.com/sp?id=kb_article_view&sys_kb_id=85cba9281b1ad950f82d9828b04bcb2d .



```
-bash-4.2$
```

Figure 2: Example shell-prompt after successfully connecting to Linux.city.ac.uk

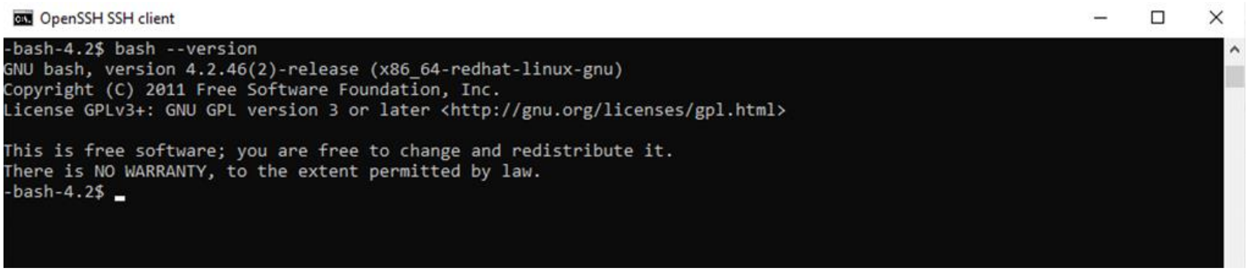
Task 1: running commands from the command-prompt

Linux, like all Unix variants, provides a variety of *command-line interfaces* (CLIs) known as *shells*. Shells allow you to use the operating system – i.e., executing applications, configuring your computer and invoking system tools. For today's tutorial, we will use the default Fedora33 console from JSLinux and the ssh (Bash) shell for City's Linux server.

Start an ssh Bash shell on City University's Linux server (see **Task 0**). Now that we are in a Linux shell let's begin executing *commands*. To execute a command we type the command at the *command prompt* and press the Enter key on the keyboard. In Figure 2, the text "`-bash-4.2$`" is the command prompt. This particular prompt is indicating that the shell we are using is an instance of *Bash*, version 4.2. The text before the "`$`" sign in the prompt may be different for you if you are using a different Linux session.

Try the `whoami` command; this command prints out your City university username, which is the username for your ssh session. To do this, type `whoami` and press the Enter key.

Many commands accept *command-line arguments* that modify the behaviour of the command. For example, you can usually check which version of a command is installed by using the "—version" modifier with the command. For example, the command `bash --version` prints out version information to the shell, telling us which version of the Bash shell we are using. At present, it is version 4.2 running on City's Linux server (see Figure 3).

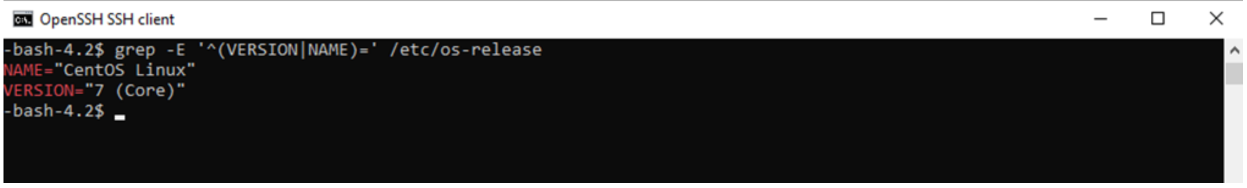


```
-bash-4.2$ bash --version
GNU bash, version 4.2.46(2)-release (x86_64-redhat-linux-gnu)
Copyright (C) 2011 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>

This is free software; you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
-bash-4.2$ _
```

Figure 3: The current version of Bash on City's Linux server

If you want to know the Linux version that is running, type `cat /etc/os-release` and press the Enter key. This prints information from the "`os-release`" file (located in the "`etc`" directory) to the shell; this information includes the name and version of Linux. "`cat`", which is short for *con**cat**enate*, is a program that reads a file it is given and prints the file contents to the shell. Another command, `grep -E '^(VERSION|NAME)=' /etc/os-release`, gives the same Linux version information more succinctly. `grep` is like `cat,` but it allows you to search the file being read for patterns, in order to print out only relevant information. Here we are using grep to search for, and print out, lines in the file that contain the strings 'VERSION=' or 'NAME=' at the beginning of the lines. If you run this command in the ssh shell, you should get output like that in Figure 4.



```
-bash-4.2$ grep -E '^(VERSION|NAME)=' /etc/os-release
NAME="CentOS Linux"
VERSION="7 (Core)"
-bash-4.2$
```

Figure 4: CentOS Linux is City University's ssh-enabled Linux server

Every command comes with a manual on how to use the command. If you come across a command you don't know how to use, issue the `man` (short for "**man**ual page") command on the command you wish to know more about. For example, for the `whoami` command's manual, try `man whoami` . To exit the man page and return to the command prompt, press the `q` key.

Question: Based on the `whoami` man page, what is an alternative command which provides the same information as `whoami`? Try out this alternative.
Activity: Connect to the ssh-enabled Linux server. The Linux version information can also be obtained using the `lsb_release` command. By using `man`, find out which command-line argument makes `lsb_release` print out a single line with the Linux distribution information. Does this information agree with the information we obtained using `grep`? Can you state the full release number for this Linux distribution?

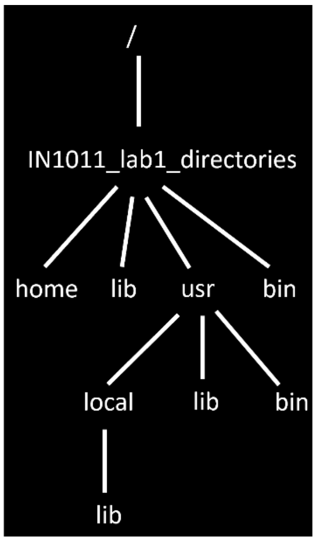


Figure 5: Directories have a tree structure with the "root" at the top

Task 2: navigating the directory structure

You will need to know about the Unix/Linux directory structure and how to navigate through directories and files. Files are stored in directories (folders) which form a tree structure.

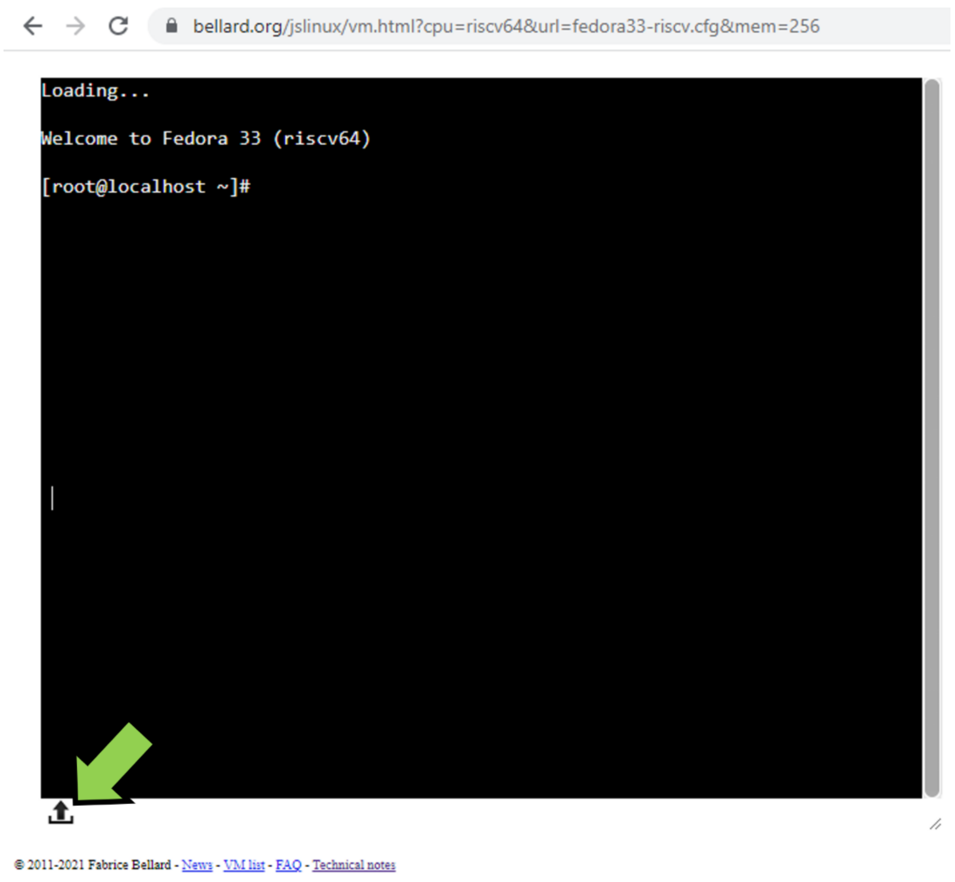
Let us create directories in a tree structure as follows. First, download the *IN1011Lab1_create_dirs.sh* and *IN1011Lab1_destroy_dirs.sh* files from the IN1011 Moodle section for this tutorial to your computer. These files are *shell scripts* – programs that perform a collection of commands to be executed in a shell. The scripts can be opened in a text editor and modified. Next, open a Linux shell in your web browser for this exercise: go to the JSLinux webpage and click on the Fedora33 console link (see Figure 1). Once the shell is open, upload the shell scripts to the JSLinux shell using the JSLinux upload button (see Figure 6). The upload will take a few seconds, during which you may not be able to type to the command prompt.

Once the command prompt becomes responsive, take a look at the files and directories in the current shell directory by issuing the command `ls -al`. You should see a list of files and directories that includes the uploaded files.

Activity: Now run the following commands:

1. `chmod 755 IN1011Lab1_create_dirs.sh` : this changes the permissions on the uploaded file, making it executable. You will see more of the `chmod` command later in IN1011;
2. `./IN1011Lab1_create_dirs.sh` : this executes the script which creates the directory structure depicted in Figure 5.

The topmost directory in Figure 5, shown by " / ", is known as the "root" directory (this tree is upside down, as is usual in Computer Science). The position of any other directory or file in the tree is called its *path*. " / " is also used as a delimiter in path expressions. For example, "/IN1011_lab1_directories/usr/lib" means: the directory called "lib", which is a child of the directory called "usr", which is a child of the directory called "IN1011_lab1_directories", which is a child of the root directory "/". Note that directory names only have to be unique within their parent directory (so, for example, there could be another directory called lib somewhere else in the tree, such as "/IN1011_lab1_directories/lib").


bellard.org/jslinux/vm.html?cpu=riscv64&url=fedora33-riscv.cfg&mem=256

```
Loading...

Welcome to Fedora 33 (riscv64)

[root@localhost ~]#
```

© 2011-2021 Fabrice Bellard - News - VM list - FAQ - Technical notes

Figure 6: The file upload button for JSLinux Fedora33 console

To delete the script-created directories, execute the IN1011Lab1_destrot_dirs.sh script. As before, you may need to make this script executable by setting its permissions using `chmod`.

Next week, in lecture 2, you will be introduced to processes. Each Linux process (e.g. an instance of the Bash shell) keeps track of "where it is" within the directory structure; i.e., the current or *working directory*. Issuing the command `pwd` prints the path of the working directory; the path is stored in a working directory variable for the shell process.

Many shell commands make use of the working directory variable. For example, the `ls` command is used to list the contents of a directory. If you enter `ls /usr/local/lib` in a shell, it will list the contents of the directory /usr/local/lib. But if you just enter `ls` by itself, it will list the contents of the current working directory. Other commands (such as `cd`) can be used to "move around" within the file system (they change the value of the working directory variable).

Activity: some important commands for working with directories are `pwd`, `cd`, `ls`, `mkdir`, `rmdir`. Find out what they do using `man`. Test your understanding by executing these commands.

Activity: create a new directory called `HatOfTheCat` inside the `IN1011_lab1_directories` directory; create some more directories inside `HatOfTheCat`. Now try to remove `HatOfTheCat`: what happens?

Paths

We distinguish between *absolute* and *relative* paths. An **absolute path** is just a full path starting from the root. For example, "/my/foo/bar" is the absolute path to the bar directory. A **relative path** shows the position of a directory *starting from some other directory*. For example, suppose that your working directory was currently "/my". Then, starting from this directory, the relative path to the bar directory is simply "foo/bar".

Some special notation used in path expressions:

- Two dots "`..`" means "parent directory". For example, the relative path of "/usr" from directory "/usr/local/lib", is "../..", because the parent of "/usr/local/lib" is "/usr/local" and the parent of "/usr/local" is "/usr".
- One dot "`.`" means "current directory". For example, "/home/troy" and "/home/troy/." mean the same thing, because the current directory starting from "/home/troy" is just "/home/troy".
- A tilde (also known as a twiddle) "~" means "the current user's home directory". For example, in a shell created by foo, "~" is equivalent to /home/foo. You can also specify a particular user. For example, "`~fred`" means fred's home directory.

Files

Similarly to directories we can show the position of a *file* in the file system. For example, in the tree above, there exist 2 empty files, `my_foo.bin` and `my_bar.bin`, located in the directory /IN1011_lab1_directories/bin. Path expressions for files are written the same way as for directories: /IN1011_lab1_directories/bin/my_foo, /IN1011_lab1_directories/bin/my_bar

Note that when a path ends with a directory (but not when it ends with a file) we can optionally write a "/" at the end of the path. For example, we could write /home/ to make sure that the reader understands that home is a directory.

Question: Assume that your current working directory is /local/lib. Now, if you execute

`cd ../../local/lib/` , what will be the absolute path of your new working directory?

Use the `cd` and `pwd` commands to check your answer.

Task 3: transferring files [note: requires a VPN connection, do NOT use JSLinux]

The file transfer application sftp provides an encrypted network connection between two machines that allows you to transfer files from one machine to the other. Open a command prompt on Windows or a terminal on Mac OSX and try the following command (replace *userid* by your City user name):

`sftp` *userid*`@linux.city.ac.uk`

When prompted for your password, enter your usual City password. If the connection succeeds, the shell command prompt will be replaced by the sftp prompt. Sftp operates like a shell but recognises a more limited, specialised set of commands. One command that it shares with Bash is the `ls` command. Try entering `ls -al` at the sftp prompt; this will display the contents of your working directory *on the remote machine*. To list the contents of the working directory on the local machine (your Windows/OSX machine) use `lls` instead of `ls`. To transfer a file from the local machine to the remote machine, use the sftp command:

`put` *[localfilename] [proposedremotefilename]*

To transfer a file in the other direction, use:

`get` *[remotefilename] [proposedlocalfilename]*

To exit from sftp, use the `exit` command.

Activity: `sftp` into the City Linux server. Use the `put` command to copy the *IN1011Lab1_create_dirs.sh*, *IN1011Lab1_destroy_dirs.sh* files from your computer to your user directory on the Linux server. Then, `ssh` into the City Linux server and execute these shell scripts in your user directory. Use the `ls` command inbetween executing these scripts, to check that directories are being created and destroyed.

Question: When *IN1011Lab1_destroy_dirs.sh* is executed to delete directories, why doesn't the error message that we got when trying to delete `HatOfTheCat` come up? Hint: open the *IN1011Lab1_destroy_dirs.sh* file in a texteditor using the `nano` command. How does the command to delete these directories differ from the usual command for deleting a single directory?