OS Shells and Command Line Programming

Edited from Stephen Mitchell - 2020

OS Shells Overview

OS Shells & The Command Line

- Definition of Shell
- What it looks like
- What can you do in a generic "Shell"

Using BASH

- History of BASH
- Opening a BASH in a Unix type system
- Simple BASH commands
- Scripting

The Labs

- Starts next week for 3 weeks, Weds and Thurs
- In the Linux Lab (CoFo 204)
- A different set of log-on credentials, you’ve been emailed your details by cosit
- Runs OpenSUSE, but you can use your own device

The Labs

- Graded labs!
- Part in-lab tasks, part assignment.
- Assignment task released in second lab week.
- Submission to Canvas, deadline: 10th April.

The Unix System

- Unix based OSs are a family of Operating System that utilise the hierarchical UNIX file system.

- Developed initially by AT&T for internal use, but it gradually spread and became more comprehensive.

- Not to be confused with a Linux kernel, which is derived from Unix but with key differences.

- Linux is an open source re-implementation of a Unix system and does not explicitly contain an OS, although distributions such as Ubuntu/OpenSUSE etc package kernel with OS.

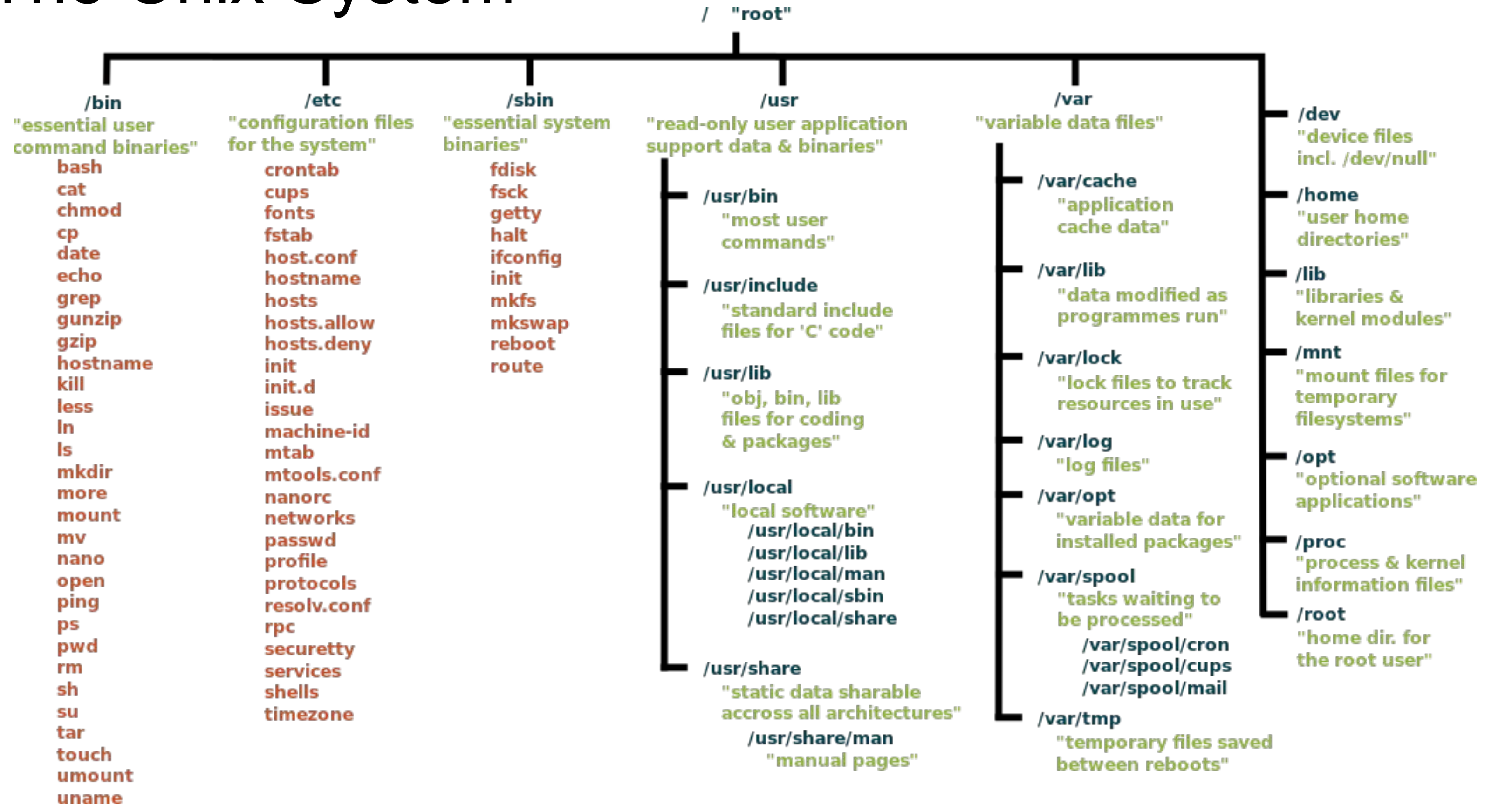
The Unix System

- A vast portion of the Unix and Unix-like system treats OS behaviours as files and manipulations of such files.

- A small number of file types are available, including:
  - Text files
  - Directories
  - Symbolic links
  - FIFO (pipes, see later)
  - …

- Many **commands** defined which interact with such file types

The Unix System



The Unix System

- As covered in previous lectures, the directory is just a special file type
  - Contains the information about all files below it.
- A file can be accessed by using it’s **relative** or **absolute** pathname
  - Absolute path describes the **path from the root**
  - Relative path describes the path in **relation to the current directory**
- Some files can be indicated as **“hidden”** by prepending their name with a period.
  - **MyFile.txt** is not hidden, but **.MyFile.txt** is.

Definition of **Shell**

- …an outer layer of an Operating System.
- …an interface between the user and the internal parts of the operating system (or the Kernel)…

[http://www.linfo.org/shell.html]

What is the Kernel?

- The kernel talks to the hardware, software and manages the systems resources (RAM, CPU, BUSes).
- A kernel is at the heart of Mac OS, Windows & Linux.
- It’s the core of the operating system and must run reliably and efficiently for the computer system to work correctly, prevent data and processes corrupting and to re-use resources efficiently.
- Linux is open source, you can download and compile your own Linux kernel if you can use the command line.

Shells

- A number of different shells have been developed for Unix-like operating systems.
- They share many similarities, but there are also some differences with regard to commands, syntax and functions that are important mainly for advanced users.
- Every Unix-like operating system has at least one built-in shell, and most have several.

http://www.linfo.org/shell.html

Shell Prompt or Command Line

- A ***shell prompt***, also referred to as a command prompt is a character or set of characters at the start of the ***command line*** that indicates that the shell is ready to receive commands.

- It usually is, or ends with, a dollar sign ($) for ordinary users and a pound sign (#) for the root (i.e., administrative) user.

- The term ***command line*** is sometimes used interchangeably with the ***shell prompt***, because that is where the user enters commands.

[http://www.linfo.org/shell.html]

The **sh** Shell

- **sh** (the Bourne Shell) is the original UNIX shell, and it is still in widespread use today.

- Written by Stephen Bourne at Bell Labs in 1974, it is a simple shell with a small size and few features, perhaps the fewest of any shell for a Unix-like operating system.

- Bell Labs was the research and development arm of AT&T (The American Telephone and Telegraph Company). The first version of UNIX was developed at Bell Labs in 1969.

http://www.linfo.org/shell.html

The ***bash*** Shell

- ***bash*** (Bourne-again shell) is the default shell on Linux.
- Runs on nearly every other Unix-like operating system as well, versions are also available for other operating systems including Windows systems.
- ***Bash*** is a superset of ***sh*** (i.e., commands that work in ***sh*** also work in ***bash***, but the reverse is not always true), and it has many more commands than ***sh***, making it a powerful tool for advanced users.

http://www.linfo.org/shell.html

Why ***bash***?

- It is intuitive and flexible, and thus it is probably the most suitable shell for beginners.

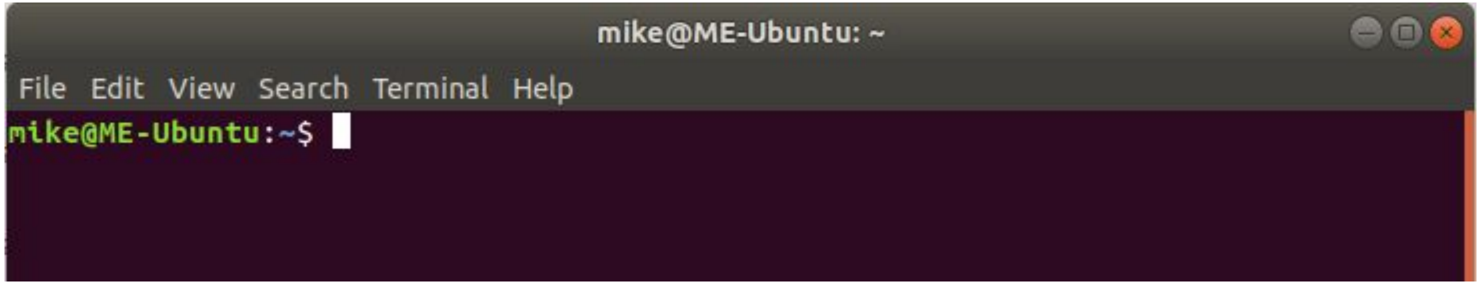
- ***bash*** was written for the GNU project (whose goal is to develop a complete, Unix-compatible, high performance and entirely free operating system), primarily by Brian Fox and Chet Ramey.

- Its name is a pun on the name of Steve Bourne.

http://www.linfo.org/shell.html

Starting directory

- Each user account will have a **home directory.**
- Contains the user's personal files and information.
- When logging into a shell, it starts within the **starting directory.**
    - This is often the user's home directory by default.



```
mike@ME-Ubuntu:~$ 
```

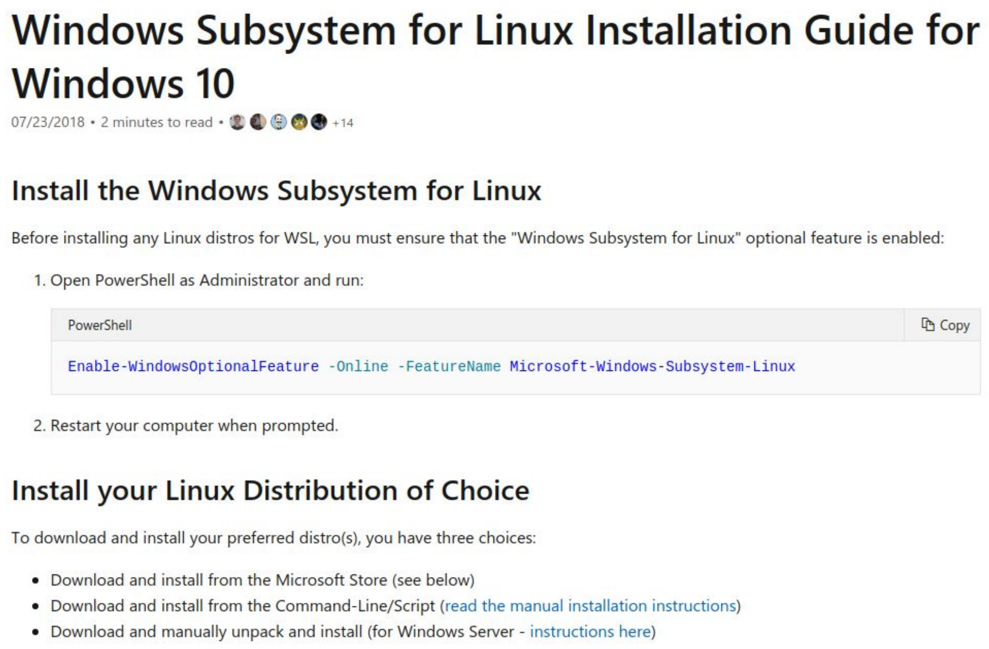
Try it for yourself now! (On laptop)

http://bit.ly/2GKDnxj

https://bellard.org/jslinux/vm.html?cpu=riscv64&url=https://bellard.org/jslinux/buildroot-riscv64.cfg&mem=256

Try it for yourself now! (On Windows 8+)

http://bit.ly/331rn2Z



Windows Subsystem for Linux Installation Guide for Windows 10

07/23/2018 • 2 minutes to read • +14

Install the Windows Subsystem for Linux

Before installing any Linux distros for WSL, you must ensure that the "Windows Subsystem for Linux" optional feature is enabled:

1. Open PowerShell as Administrator and run:

PowerShell

Copy

```
Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux
```

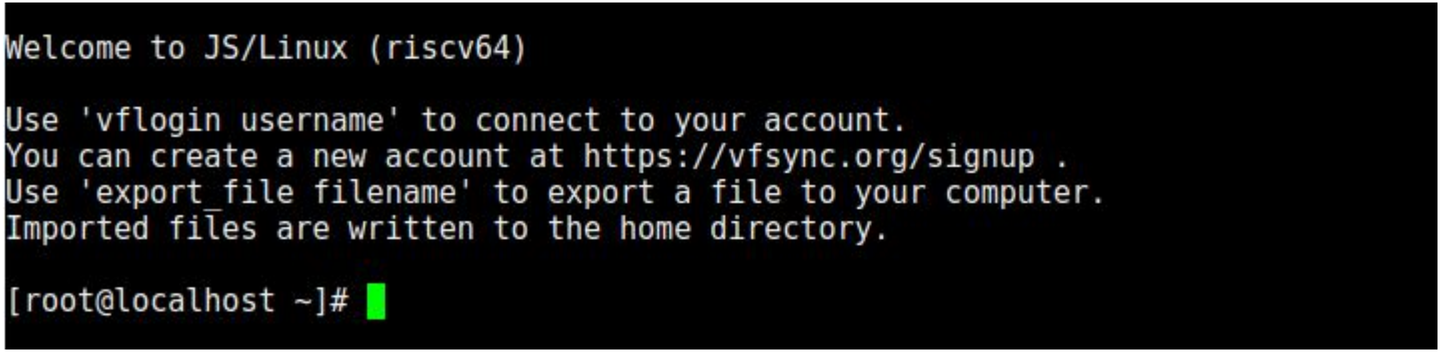
2. Restart your computer when prompted.

Install your Linux Distribution of Choice

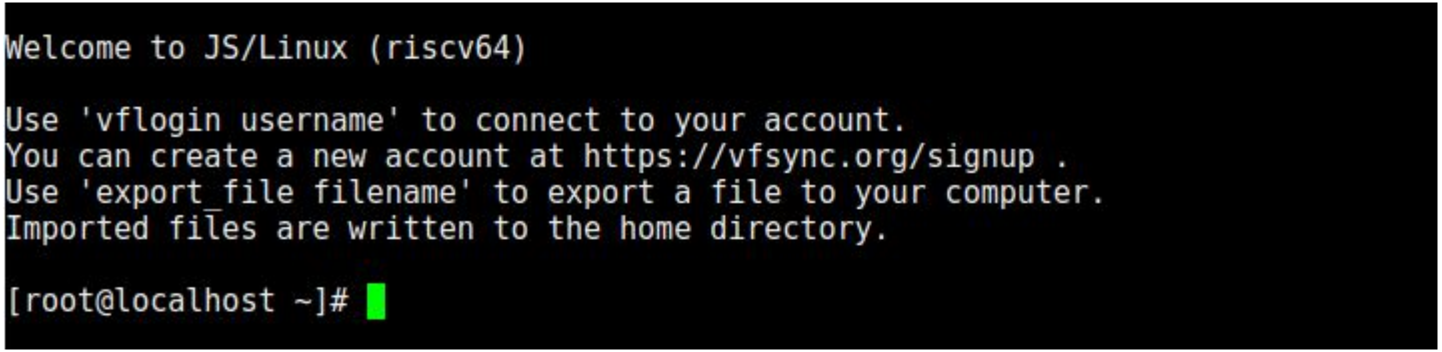
To download and install your preferred distro(s), you have three choices:

- Download and install from the Microsoft Store (see below)
- Download and install from the Command-Line/Script (read the manual installation instructions)
- Download and manually unpack and install (for Windows Server - instructions here)

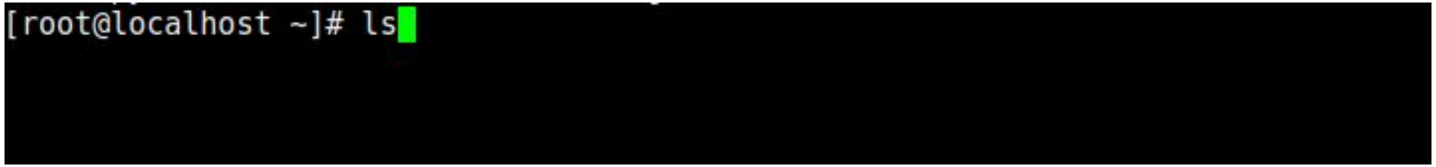
https://docs.microsoft.com/en-us/windows/wsl/install-win10



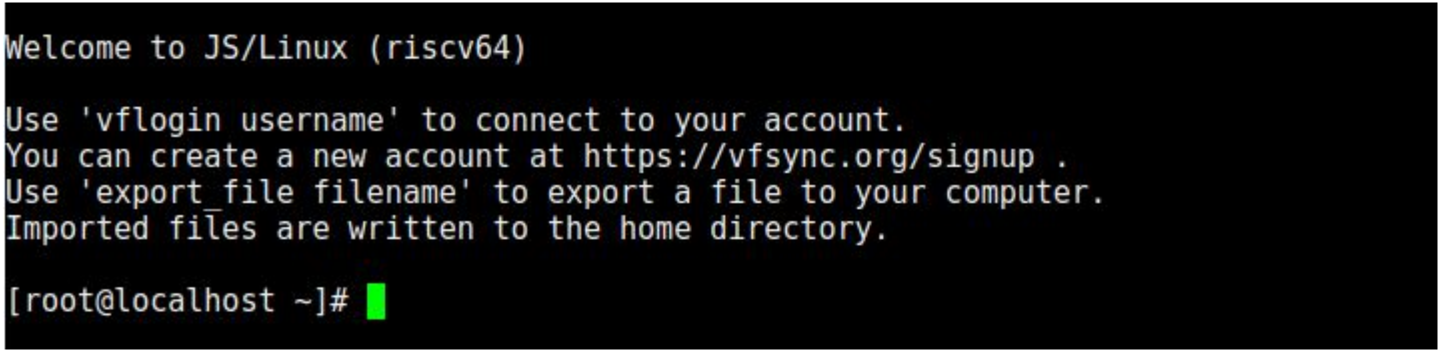
```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# 
```



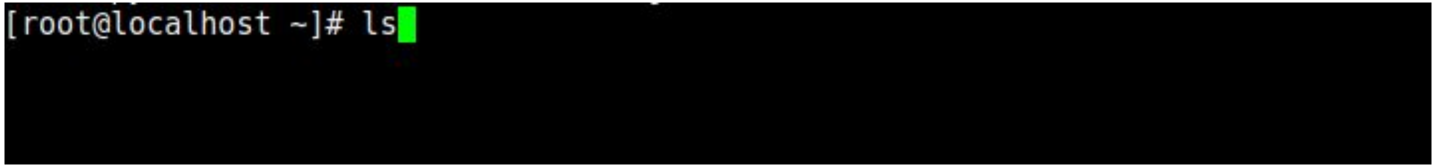
```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]#
```



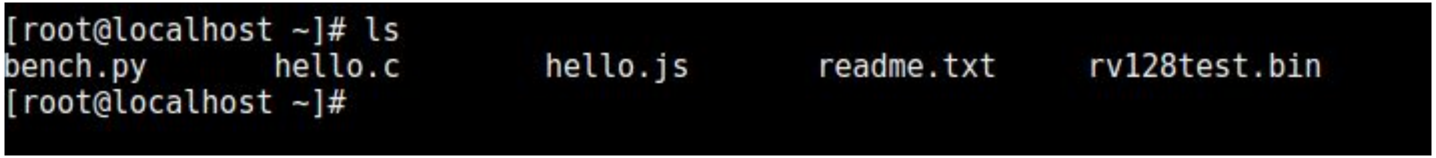
```
[root@localhost ~]# ls
```



```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]#
```



```
[root@localhost ~]# ls
```



```
[root@localhost ~]# ls
bench.py       hello.c         hello.js       readme.txt     rv128test.bin
[root@localhost ~]#
```

Evolution of the Interface

- User Interface was carried out using a series of commands executed in a batch, often fed in using punch cards or tape.
- These become known as ‘interfaces’.
- Replaced by batch files or scripts today and run from the command line or shell.

GUI & CLI

- GUI (Graphical User Interface) provides a visual user experience and needs a navigation device such as a mouse or touch screen (although you can tab around some options).
- If you are used to using a mouse or a touch screen and are familiar with looking at menus and icons a GUI will let the user explore an OS's functionality without the need to know key command words or any command syntax.
- CLI (Command Line Interface) uses a text interface only.
- The CLI can be connected to with just a keyboard and text output, allowing access to a computer's OS interface without needing a screen.

Commands and Applications

- A shell is a command-line interpreter, it is designed to execute programs called **commands**.
- It can also execute larger programs, applications and user-defined code.
- Commands come in the following form:

$ command [arg1] [arg2] …[argn]

Where $ is the command prompt, and [arg] indicates a number of optional arguments passed to the command.

Commands and Applications

- The Unix OS family is known for being terse (concise)
- Commands can often be short and cryptic, with little use of vowels
    - ls: **L**i**s**t
    - cd: **C**hange **D**irectory
    - grep: **G**lobally search a **R**egular **E**xpression and **P**rint
- Not always helpful / memorable. But you'll become proficient with practice.

Commands and Applications

- Many commands have optional flags or option switches.
- In the same vein as being concise, and avoiding a large number of very similar commands with slightly varied behaviour.
- Flags are often (but not always!) denoted with a - or -- notation and used to indicate functionality to carry out.
- Although they have this special behaviour, they are just arguments passed to the command as seen in the last slide.

Commands and Applications

- Consider the command **ls**
- Has flags/options/switches which allow longlisting and recursion
- These are both the same:

$ ls **-l -R** ./Documents

$ ls **-lR** ./Documents

$ ls **-l --recursive** ./Documents

- Why does -l not have a -- form?

Built-in Commands vs External Executables

- A shell provides an environment in which we can run our available commands.

- In order to do so we require a number of built-in commands which ship with the shell

- These can vary per shell, but include the basics like:
  - cd - Changing directory
  - exec - Executing an external executable command
  - pwd - Print absolute path of current directory

- BASH also defines its own built-ins. Can you find which are BASH only?

Built-in Commands vs External Executables

- Built in commands such as **clear** (clear screen), **cd** (change directory) and **mkdir** (make directory) are present in the shell.

- Applications such as **nano** (a text editor), can be installed started in the command line.

- Services such as **httpd** (apache web server), **ftp** (files transfer protocol) and **ssh** (secure shell) can be started and stopped using the Linux command line.

Built-in Commands vs External Executables

- Calling a built-in command runs directly in the shell.
- Calling an external executable requires the program to be loaded and executed. This is inherently slower.
- Some functionality can actually behave very differently if not defined as a built-in.
    - Will cd work as an external executable program?

Some basic commands to get you going

mkdir - Make a directory

> ***mkdir mydirectory*** - make directory in current directory which is called “mydirectory”

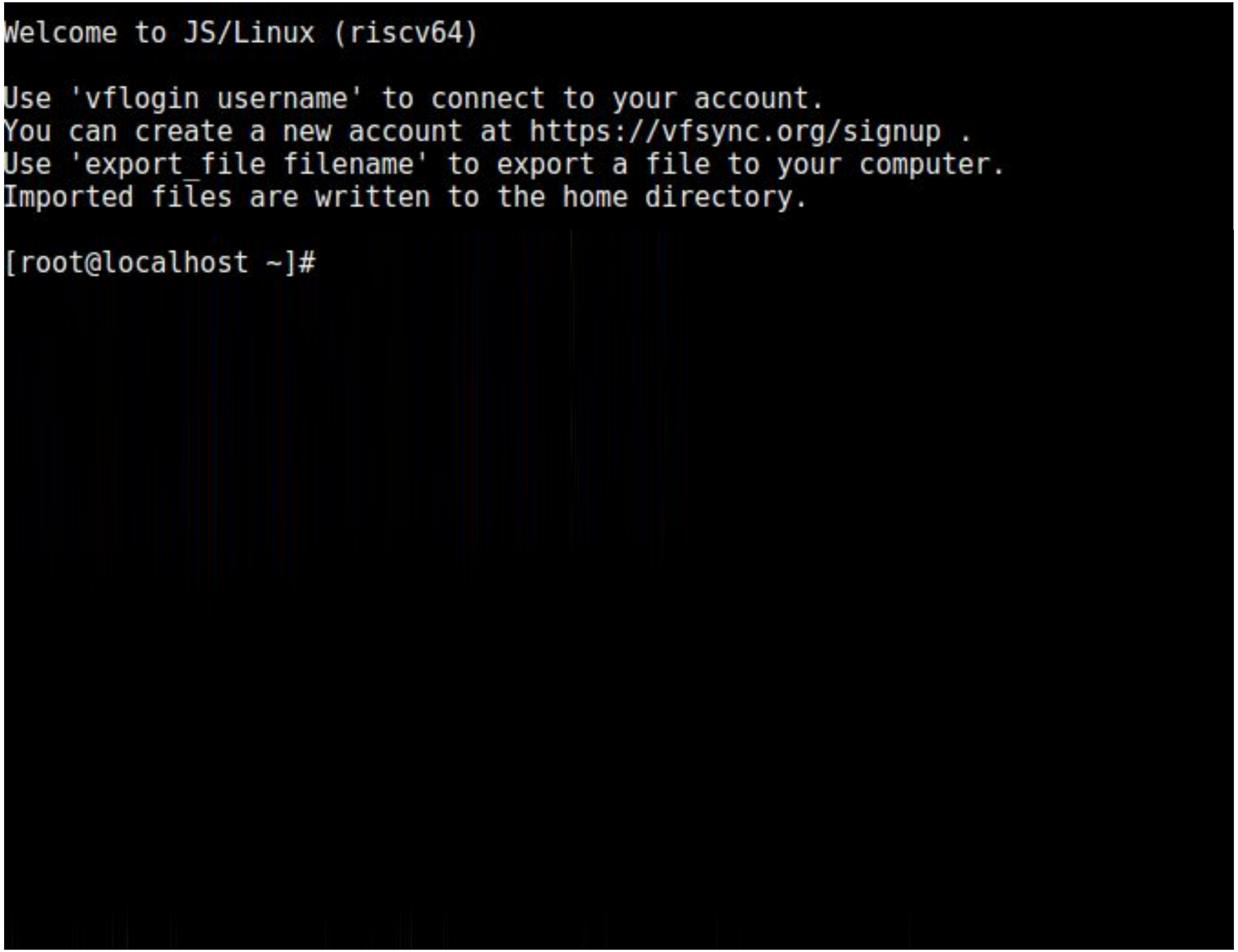
cd - Change Directory

> ***cd mydirectory*** - change to “mydirectory” **iff** it exists
>
> ***cd ..*** - change directory into the parent directory

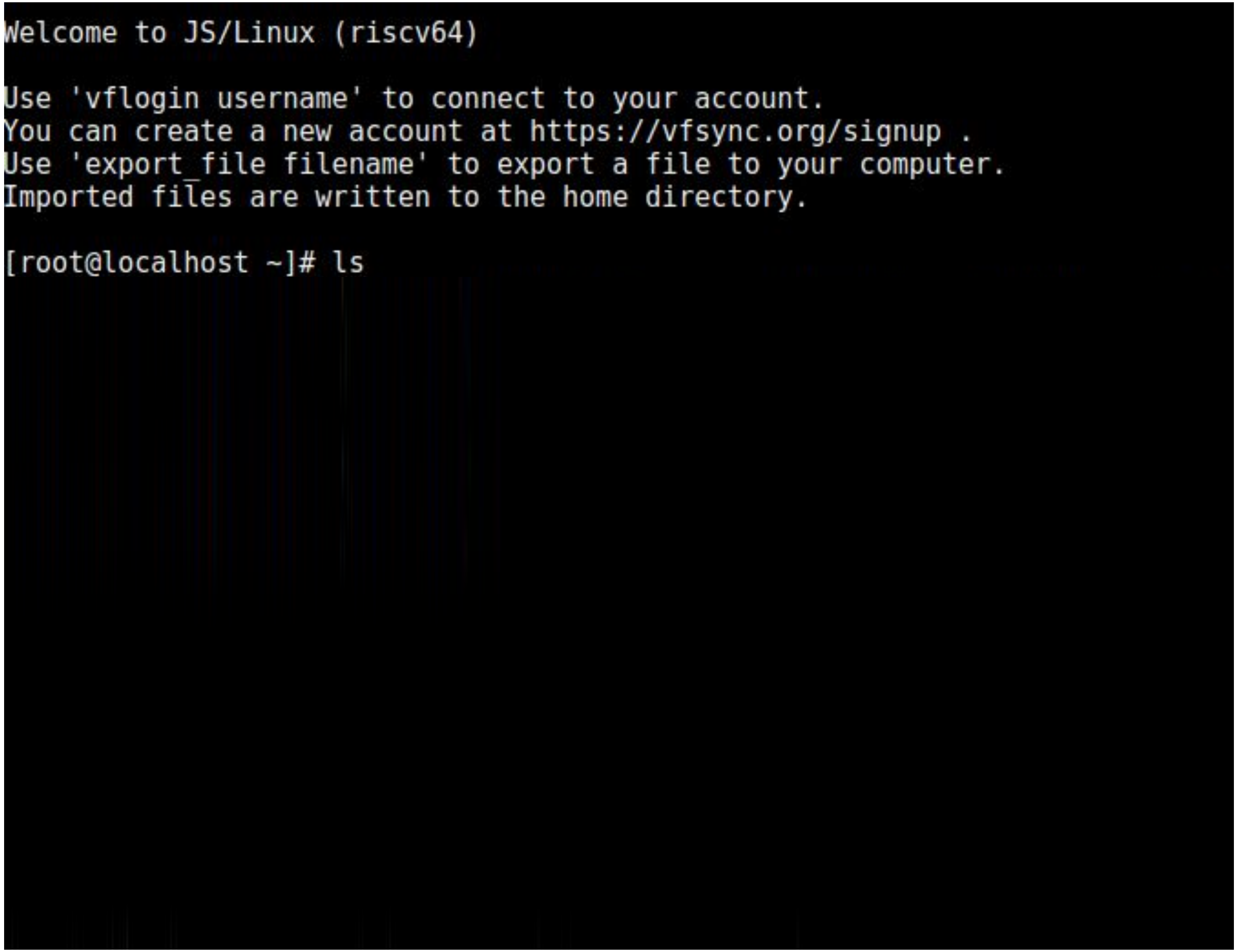
Some basic commands to get you going

ls - List directory contents
- **ls** - list the current directory's visible contents
- **ls mydirectory** - list visible contents of "mydirectory"
- **ls -a mydirectory** - list visible & hidden contents of "mydirectory"

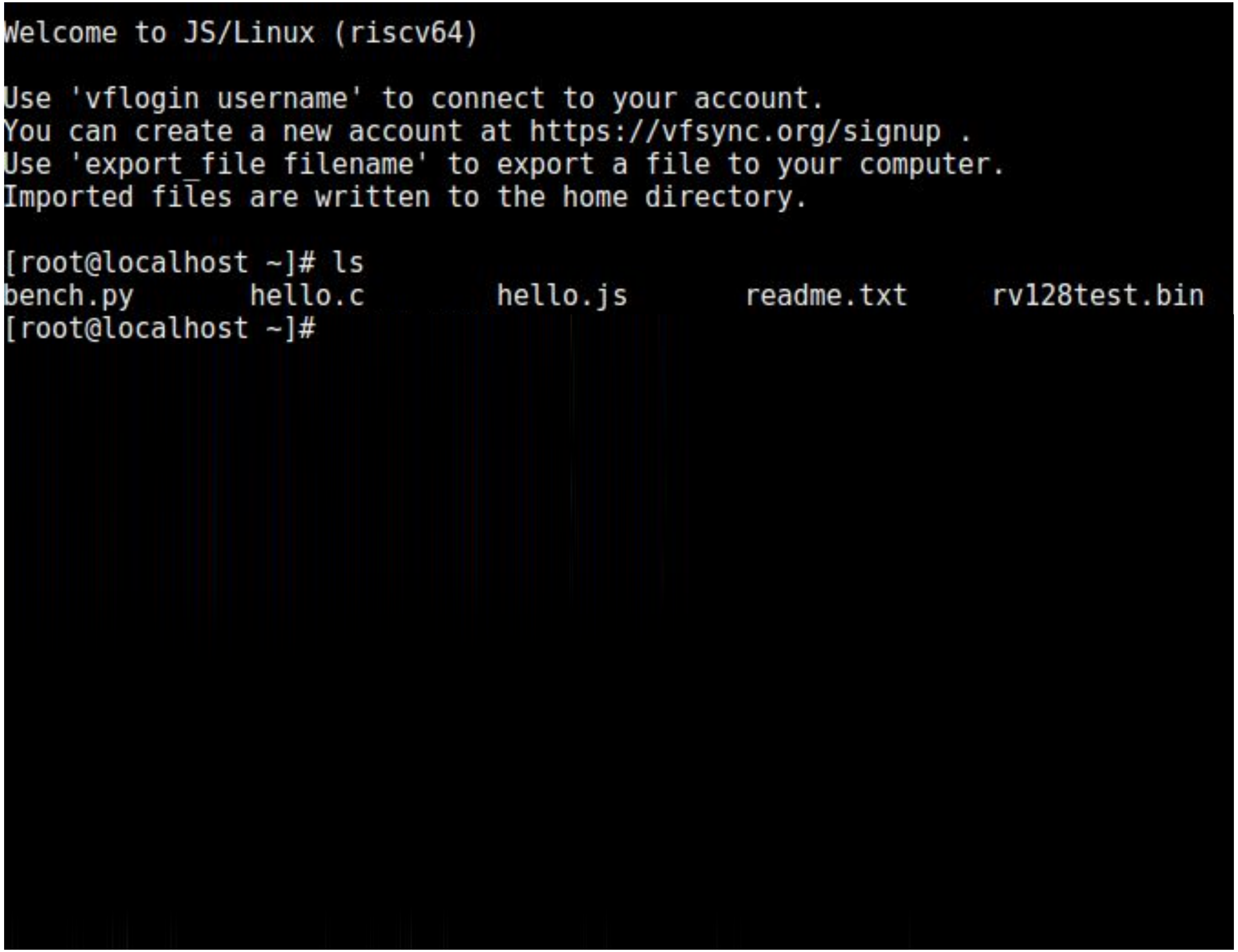
touch - Create file if it doesn't exist, or update last access time.
- **touch myfile** - creates "myfile" file if it doesn't already exist
- **touch myfile** - updates time last accessed



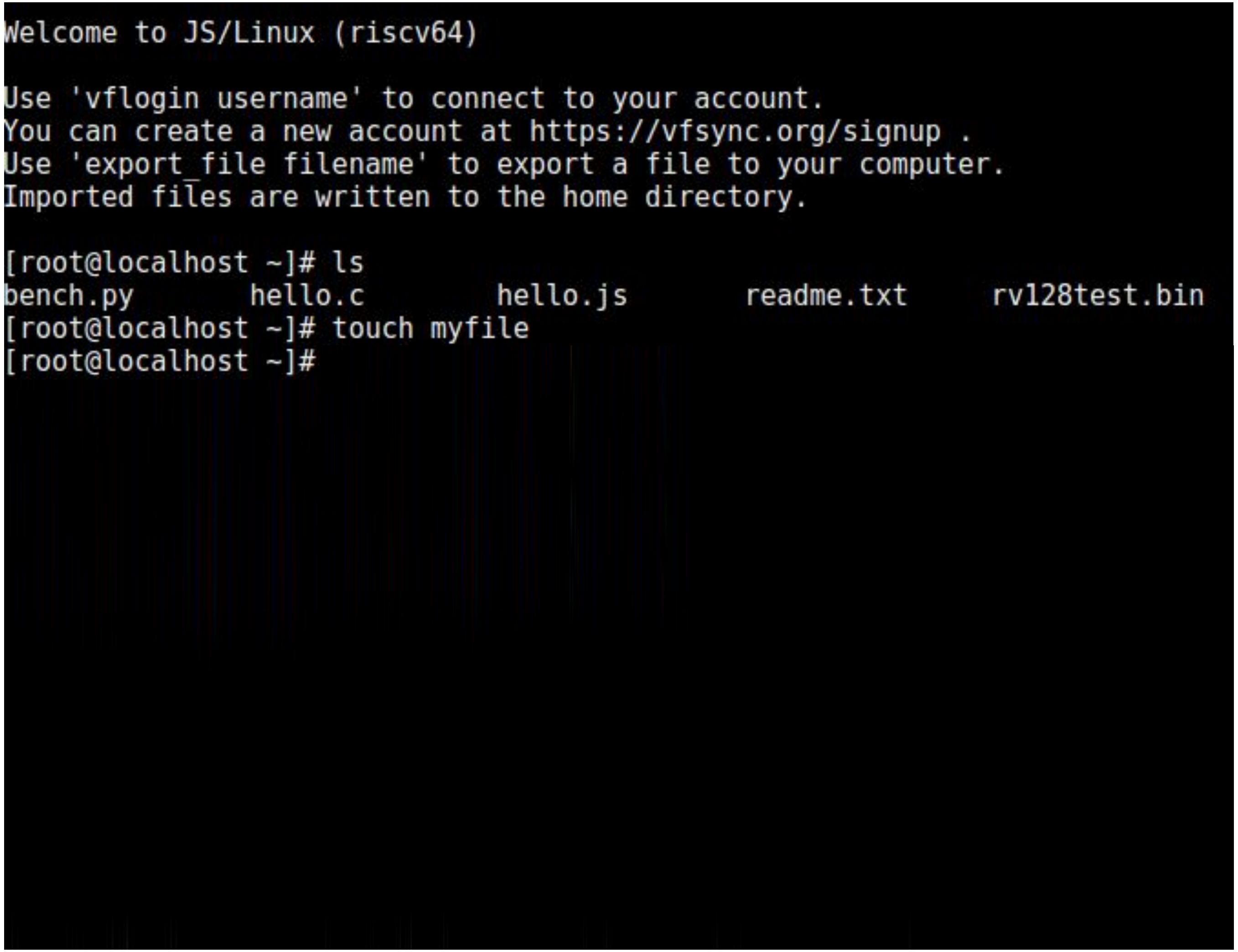
```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]#
```



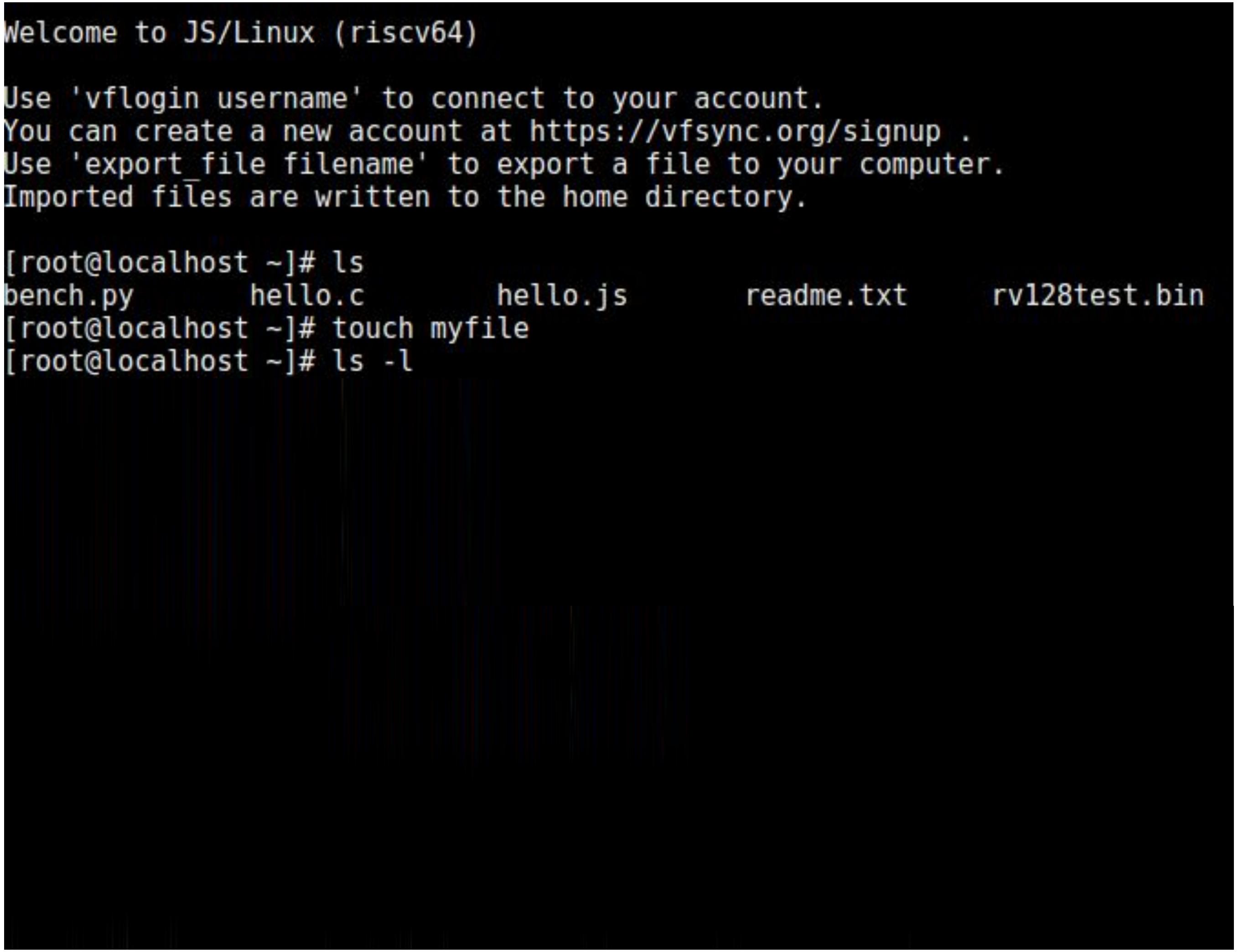
```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# ls
```



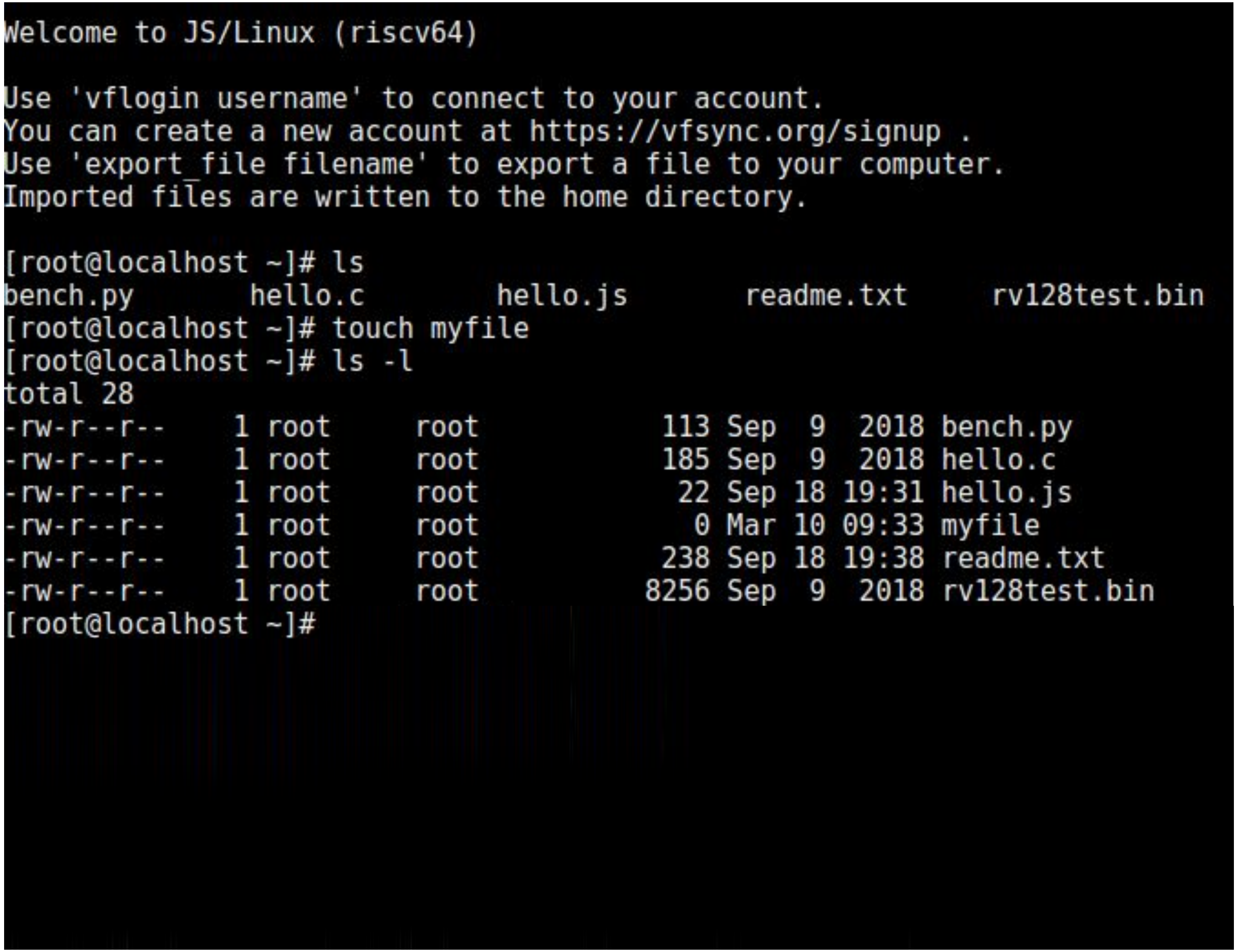
```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# ls
bench.py       hello.c        hello.js       readme.txt     rv128test.bin
[root@localhost ~]#
```



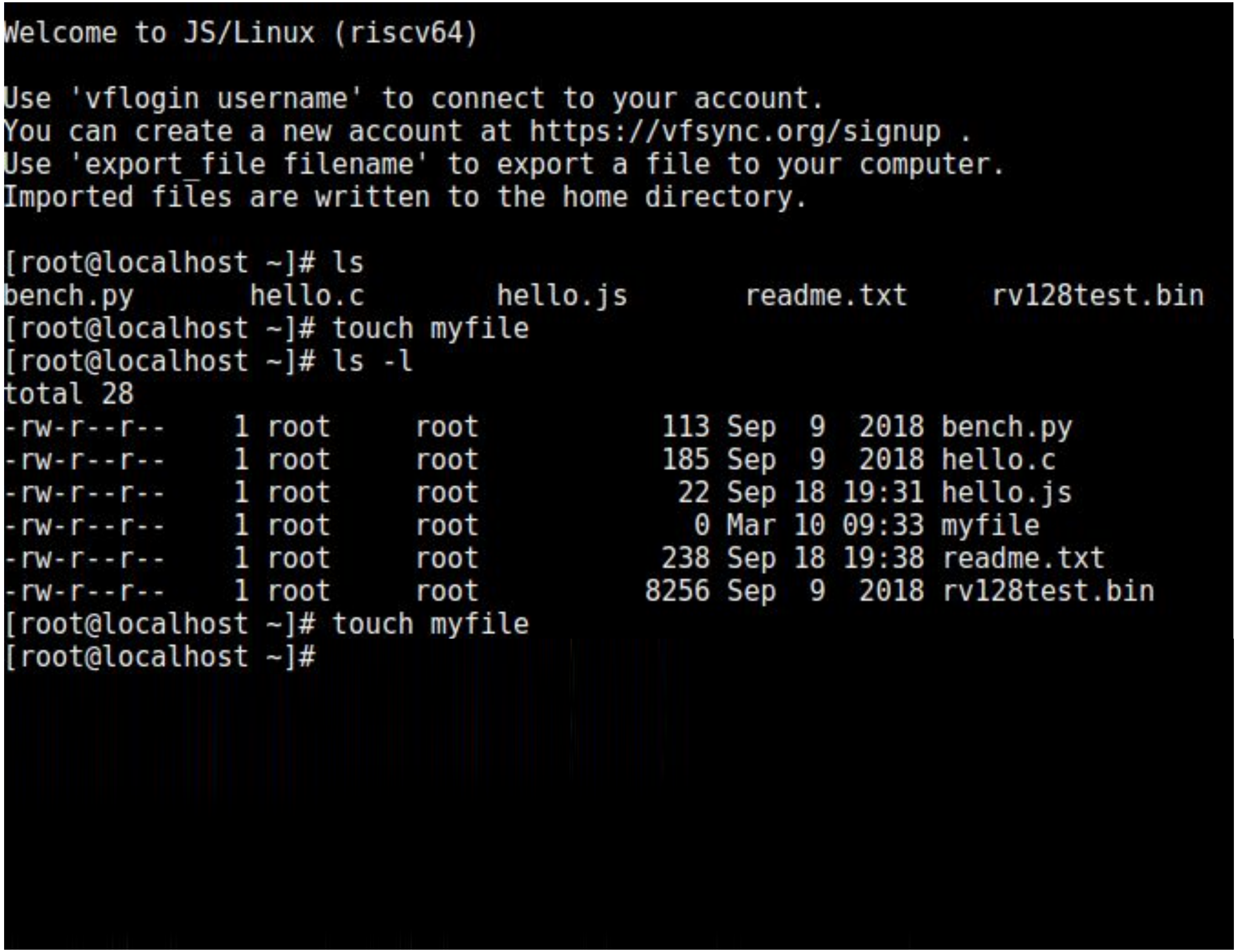
```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# ls
bench.py       hello.c        hello.js       readme.txt     rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]#
```



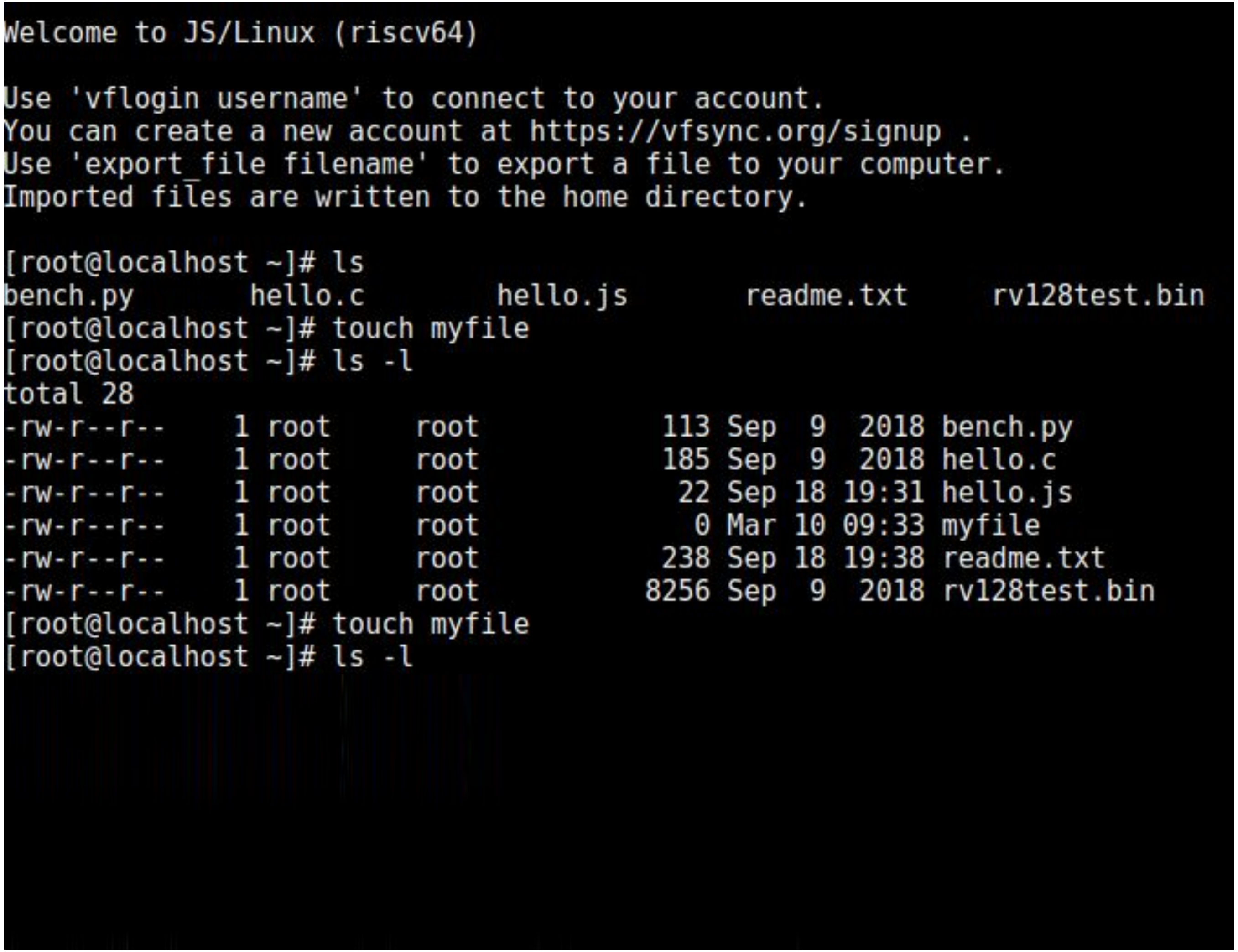
```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# ls
bench.py       hello.c         hello.js       readme.txt     rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]# ls -l
```



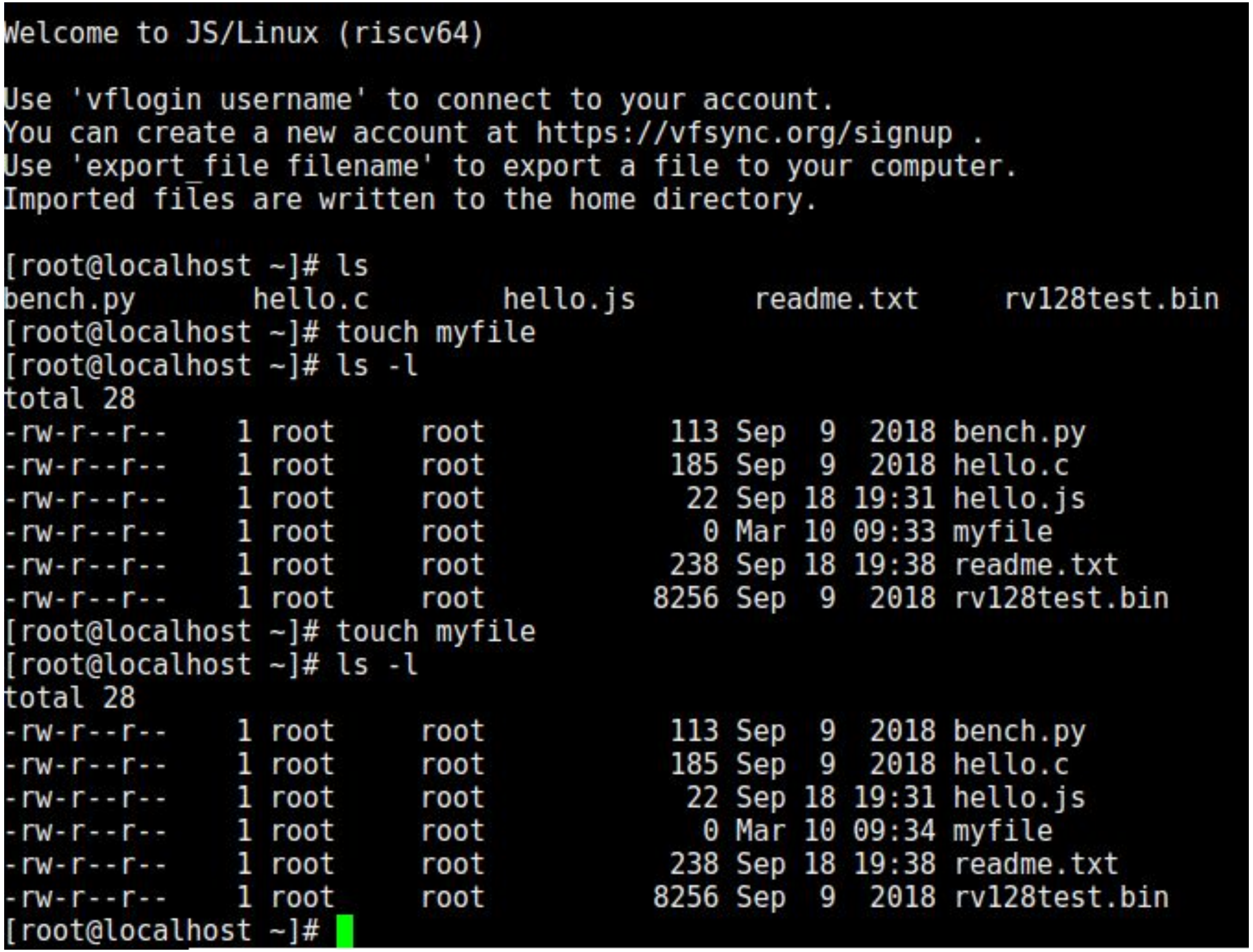
```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# ls
bench.py       hello.c        hello.js       readme.txt     rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]# ls -l
total 28
-rw-r--r--    1 root     root           113 Sep  9  2018 bench.py
-rw-r--r--    1 root     root           185 Sep  9  2018 hello.c
-rw-r--r--    1 root     root            22 Sep 18 19:31 hello.js
-rw-r--r--    1 root     root             0 Mar 10 09:33 myfile
-rw-r--r--    1 root     root           238 Sep 18 19:38 readme.txt
-rw-r--r--    1 root     root          8256 Sep  9  2018 rv128test.bin
[root@localhost ~]#
```



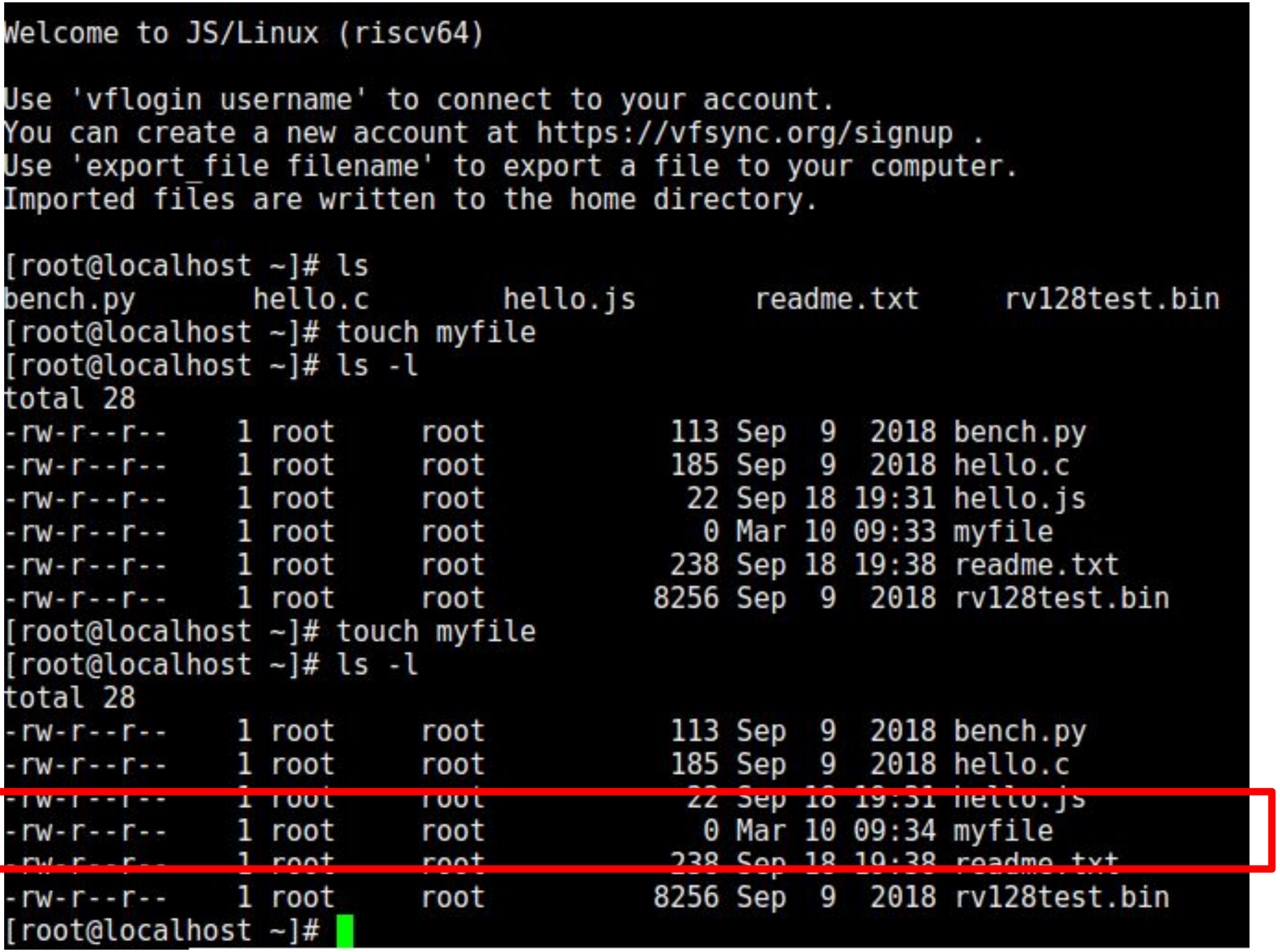
```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# ls
bench.py       hello.c        hello.js       readme.txt     rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]# ls -l
total 28
-rw-r--r--    1 root     root           113 Sep  9  2018 bench.py
-rw-r--r--    1 root     root           185 Sep  9  2018 hello.c
-rw-r--r--    1 root     root            22 Sep 18 19:31 hello.js
-rw-r--r--    1 root     root             0 Mar 10 09:33 myfile
-rw-r--r--    1 root     root           238 Sep 18 19:38 readme.txt
-rw-r--r--    1 root     root          8256 Sep  9  2018 rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]#
```



```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# ls
bench.py       hello.c        hello.js       readme.txt     rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]# ls -l
total 28
-rw-r--r--    1 root     root           113 Sep  9  2018 bench.py
-rw-r--r--    1 root     root           185 Sep  9  2018 hello.c
-rw-r--r--    1 root     root            22 Sep 18 19:31 hello.js
-rw-r--r--    1 root     root             0 Mar 10 09:33 myfile
-rw-r--r--    1 root     root           238 Sep 18 19:38 readme.txt
-rw-r--r--    1 root     root          8256 Sep  9  2018 rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]# ls -l
```



```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# ls
bench.py       hello.c        hello.js       readme.txt     rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]# ls -l
total 28
-rw-r--r--    1 root     root           113 Sep  9  2018 bench.py
-rw-r--r--    1 root     root           185 Sep  9  2018 hello.c
-rw-r--r--    1 root     root            22 Sep 18 19:31 hello.js
-rw-r--r--    1 root     root             0 Mar 10 09:33 myfile
-rw-r--r--    1 root     root           238 Sep 18 19:38 readme.txt
-rw-r--r--    1 root     root          8256 Sep  9  2018 rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]# ls -l
total 28
-rw-r--r--    1 root     root           113 Sep  9  2018 bench.py
-rw-r--r--    1 root     root           185 Sep  9  2018 hello.c
-rw-r--r--    1 root     root            22 Sep 18 19:31 hello.js
-rw-r--r--    1 root     root             0 Mar 10 09:34 myfile
-rw-r--r--    1 root     root           238 Sep 18 19:38 readme.txt
-rw-r--r--    1 root     root          8256 Sep  9  2018 rv128test.bin
[root@localhost ~]# 
```



```
Welcome to JS/Linux (riscv64)

Use 'vflogin username' to connect to your account.
You can create a new account at https://vfsync.org/signup .
Use 'export_file filename' to export a file to your computer.
Imported files are written to the home directory.

[root@localhost ~]# ls
bench.py       hello.c        hello.js       readme.txt     rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]# ls -l
total 28
-rw-r--r--    1 root     root           113 Sep  9  2018 bench.py
-rw-r--r--    1 root     root           185 Sep  9  2018 hello.c
-rw-r--r--    1 root     root            22 Sep 18 19:31 hello.js
-rw-r--r--    1 root     root             0 Mar 10 09:33 myfile
-rw-r--r--    1 root     root           238 Sep 18 19:38 readme.txt
-rw-r--r--    1 root     root          8256 Sep  9  2018 rv128test.bin
[root@localhost ~]# touch myfile
[root@localhost ~]# ls -l
total 28
-rw-r--r--    1 root     root           113 Sep  9  2018 bench.py
-rw-r--r--    1 root     root           185 Sep  9  2018 hello.c
-rw-r--r--    1 root     root            22 Sep 18 19:31 hello.js
-rw-r--r--    1 root     root             0 Mar 10 09:34 myfile
-rw-r--r--    1 root     root           238 Sep 18 19:38 readme.txt
-rw-r--r--    1 root     root          8256 Sep  9  2018 rv128test.bin
[root@localhost ~]#
```

Batch Commands or Scripting

- Command-line programming is powerful, but writing out a larger process sequentially every time can be exhausting.
- Often we may combine multiple command calls into a single script or batch.
- A series of shell commands executed from a file.
- Called **.bat** files in Windows type OS or **shell script** files in Unix / Linux.

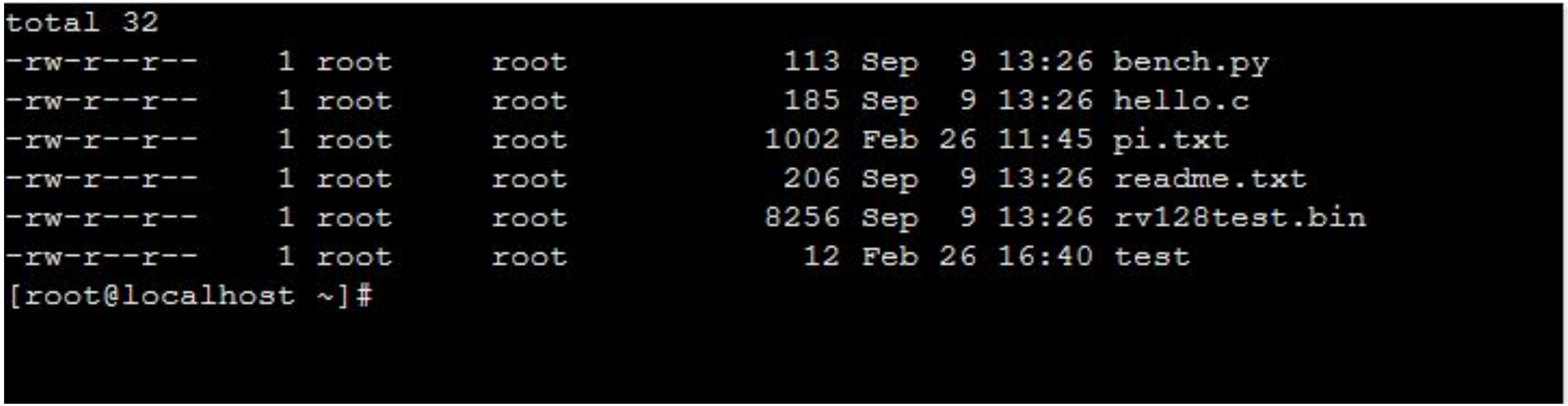
Batch ***bash*** Command using ***sh***

- You can execute a text file as a script using the ***sh*** command.

```
# nano test


<CTRL> O
<ENTER>
<CTRL> X
# sh test
```

```
clear
ls -l
```

Example Output “sh test”



```
total 32
-rw-r--r--    1 root     root             113 Sep  9 13:26 bench.py
-rw-r--r--    1 root     root             185 Sep  9 13:26 hello.c
-rw-r--r--    1 root     root            1002 Feb 26 11:45 pi.txt
-rw-r--r--    1 root     root             206 Sep  9 13:26 readme.txt
-rw-r--r--    1 root     root            8256 Sep  9 13:26 rv128test.bin
-rw-r--r--    1 root     root              12 Feb 26 16:40 test
[root@localhost ~]#
```

Bonus Material

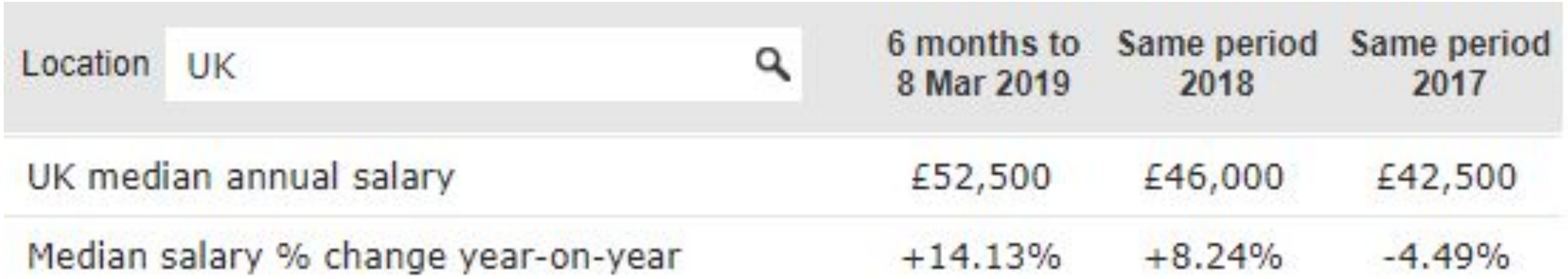
- How to Compile Your Own Linux Kernel
- Comparative Developer Salaries

How to Compile Your Own Linux Kernel

Austin Luong March 9, 2017

further reading:

https://www.makeuseof.com/tag/compile-linux-kernel/



Location UK	6 months to 8 Mar 2019	Same period 2018	Same period 2017
UK median annual salary	£52,500	£46,000	£42,500
Median salary % change year-on-year	+14.13%	+8.24%	-4.49%

Selected lines taken from on 8/3/2019
https://www.itjobswatch.co.uk/jobs/uk/linux%20command%20line.do

Comparative developer salaries (US).

Linux Kernel Developer	$162,595
Python Developer	$128,412
Swift Developer	$113,696
Java Developer	$105,888
Software Developer	$86,355
Web Developer	$72,229

From: ziprecruiter.com/Salaries Feb 2020

Comparative developer salaries.

Security Engineer	
Data Science Engineer	
Software Engineer	
IT Programme Manager	

From: IT Jobs Watch Jan 2020

Comparative developer salaries.

Security Engineer	£60,000
Data Science Engineer	
Software Engineer	
IT Programme Manager	

From: IT Jobs Watch Jan 2020

Comparative developer salaries.

Security Engineer	£60,000
Data Science Engineer	£70,000
Software Engineer	
IT Programme Manager	

From: IT Jobs Watch Jan 2020

Comparative developer salaries.

Security Engineer	£60,000
Data Science Engineer	£70,000
Software Engineer	£55,000
IT Programme Manager	

From: IT Jobs Watch Jan 2020

Comparative developer salaries.

Security Engineer	£60,000
Data Science Engineer	£70,000
Software Engineer	£55,000
IT Programme Manager	£77,500

From: IT Jobs Watch Jan 2020

Comparative developer salaries.

Security Engineer	£60,000
Data Science Engineer	£70,000
Software Engineer	£55,000
IT Programme Manager	£77,500
Computer Science Lecturer	

From: IT Jobs Watch Jan 2020

Comparative developer salaries.

Security Engineer	£60,000
Data Science Engineer	£70,000
Software Engineer	£55,000
IT Programme Manager	£77,500
Computer Science Lecturer	~£40,000

From: IT Jobs Watch Jan 2020

Comparative developer salaries.

Security Engineer	£60,000
Data Science Engineer	£70,000
Software Engineer	£55,000
IT Programme Manager	£77,500
Computer Science Lecturer	~£40,000 😭😭😭😭😭😭

From: IT Jobs Watch Jan 2020

Some ***bash*** Commands, in use.

- **The 10 Most Important Linux Commands** **http://www.informit.com/blogs/blog.aspx?uk=The-10-Most-Important-Linux-Commands** [Brad Yale, informit.com]

- **ls [list], cd [change directory], mv [move],**
- **man [manual],**
- **mkdir [make new directory], rmdir [remove directory],**
- **touch [make file], rm [remove file],**
- **locate [find file],**
- **clear [clears the screen].**

http://bit.ly/2VwoQsF

bash Reference Material

- https://ss64.com/bash/ : a handy alphabetical list of commands
- Directory Commands
- File Commands
- Redirection
- Common Flags
- Permissions
- Wild Cards
- Process Control

Useful links to Bash tutorials and explanations:

https://www.learnshell.org/
Interacting BASH scripting tool. (Not great in my opinion, but some interesting examples).

https://guide.bash.academy/inception/
Nice explanations of what BASH is and how it works.

https://linuxconfig.org/bash-scripting-tutorial-for-beginners
Some short videos of the tools working, less wordy than guide.bash.academy

https://ryanstutorials.net/bash-scripting-tutorial/
Some good recommendation online for this tutorial. Odd font for headings and several ads, but worth a look.