Computer System Internals – workshop tasks for week 1

Introduction to Ubuntu

Please remember that a practical workshop is not a race to get finished. Work systematically through the handout, making notes as you go, to make sure you understand why I have asked you to do the things I have, and why the computer is behaving in the way that it is. Remember that if you make a very small typing mistake in a command-line command that the computer may well not understand what you want it to do. So if you think it has not understood then look again at what you typed and compare it with what I asked you to type. Ask me if you are not sure.

If you finish early, go back and try some things again. Or explore and try to find new things.

If you don't finish in class time then come back when you have a free period and keep going (free periods are very useful for this purpose, and students who learn to make effective use of them will be very successful). Next week's workshop will assume you have finished this one.

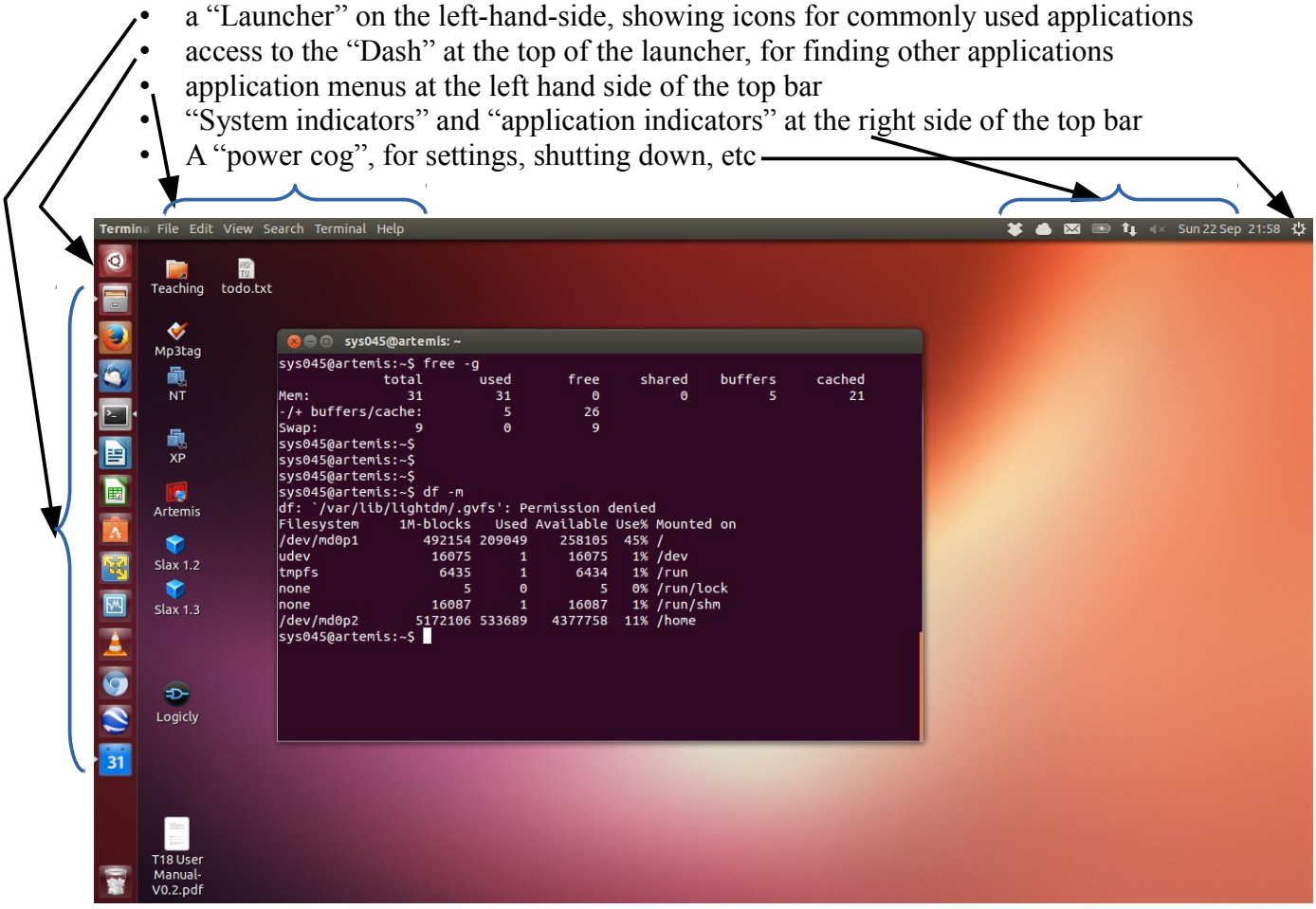
The aim of this workshop is to get everyone to boot the computers in the Lab into Ubuntu Linux and to have a look around. One particular aim is for people to have an initial look at the BASH command prompt, which we will look at in much more detail in future workshops.

When you first turn the Lab 140/141/142/149/242/245 computers on, you should get a boot menu, asking you to choose between Windows and Linux. Choose Linux.

Then use your normal university login and password to log into Linux.

When the login is completed, you will see Ubuntu's "Unity" graphics user interface. This has

- a "Launcher" on the left-hand-side, showing icons for commonly used applications
- access to the "Dash" at the top of the launcher, for finding other applications
- application menus at the left hand side of the top bar
- "System indicators" and "application indicators" at the right side of the top bar
- A "power cog", for settings, shutting down, etc



The aim of the the graphical user interface is to make the system as usable as Windows or MacOS, and to provide an interface that works on netbooks and tablets. It's a work-in-progress, and the nature of Linux is that it probably always will be.

Explore the launcher and the indicators to see what you can find.

A look at the Command Line

Although graphical user interfaces are used to provide interactive applications that have a high degree of usability and functionality, there are still many places where using a command line is easier, or is more powerful, or is the only facility available.

- Repetitive tasks or periodic tasks
- Applying an operation to a large number of objects
- Providing a simple means of customising a system
- For system administration tasks, it is often quicker
- If I am are logged in to a computer remotely (i.e. over the network) then the command line may be the only way I have to interact with the computer (for example I can log into the firstyearmatters.info server from my phone, so I can administer it while I'm on the train – but I only have access to a command-line interface).
- For system tasks that require high levels of system access, use of a command line makes it easy to give that privilege only where it is needed (with a graphical application, it is likely the whole application would need to be given privileges – consider the way Windows 7 asks for an administrator password when a system-level operation is needed by an application)
- Servers or routers often don't have GUI software installed – on the grounds that it is very complex and error-prone software that is not relevant to the core functionality and therefore that increases load and reduces stability unnecessarily.

One aim I will have throughout this module is that everyone will get some familiarity (and some confidence) in doing simple operations using a command line. We will have one workshop a week. I will start assuming no prior command line use, and I hope we'll make some progress.

Starting a "Terminal"

There are three ways to start a terminal (the equivalent of a "Windows Command Prompt")

The first way is to press `Alt-Ctrl-T`. A window appears with a prompt and a cursor, waiting for you to type in a command. Now we've seen it, let's get rid of it. Type in the command

```
exit
```

and press Enter. The window will disappear. The `exit` command tells the terminal we've finished.

The second way is to click on the "Dash Home" Ubuntu icon at the top of the launcher. The "Dash" screen will appear. Type "`terminal`" into the search field at the top of the screen. One of the applications it will find will be "Terminal". Click on that and the terminal window will appear.

Let's get rid of it a different way. Press "`ctrl-D`" and the window will disappear. Control-D is the Linux way of saying "end of file". You have told the terminal that there is no more input, so it closes.

The third way is to add an icon to the Launcher. Click on the "Dash Home" Ubuntu icon at the top of the launcher. The "Dash" screen will appear. Type "`terminal`" into the search field at the top of the screen. One of the applications it will find will be "Terminal". Rather than clicking on it, this time you should drag it onto the launcher. When you release the mouse button, the icon should add itself to the launcher. Then all you need to do to launch a terminal is to click on the icon on the launcher.

Note: please do **not** get in the habit of using the red ⊗ at the top left of the terminal window to close the terminal. Always use the `exit` command.

It is always the case that

- if you know of a keyboard shortcut then it is the quickest way to do something
- if you don't know what the shortcut is, or perhaps you don't know what the command is, then the "Dash" gives you a way of finding things (and that sometimes includes finding things you didn't know were there!)

How did I know the keyboard shortcut was `Alt-Ctrl-T`? Click on the "cog" icon at the top-right of the screen, and choose "`System Settings`", then "`Keyboard`" and then go to the "`Shortcuts`" tab. You will find details of lots of shortcuts (and there is the facility to add new shortcuts of your own).

Let's do something

However you do it, start up a terminal. As I said before, you have

- a prompt (for me it says `sys045@artemis:~$`)
- a cursor (a solid square).

This prompt and cursor is because the terminal is running a **shell**. A shell is a program that waits for you to type in a command line, works out what you want to do, and does it for you (if it can, or gives you an error message if it can't). The shell we use is called "BASH" ("Bourne Again Shell" - called this because it is an improved version of Stephen Bourne's "Bourne Shell" which was part of early versions of Unix).

Let's find out some things about the **artemis** server (this is the name of the server I am logged into – you are logged into your local PC so your numbers will be different). Type the command

`free`

and press return. You will get some output, looking something like

```
sys045@artemis:~$ free
             total       used       free     shared    buffers     cached
Mem:      32944672   32677108     267564          0    5336356   22031576
-/+ buffers/cache:    5309176   27635496
Swap:     10239464          0   10239464
sys045@artemis:~$
```

What does all this mean? Well it is something to do with the server's RAM memory. The "Total"/"Used"/"Free" headings may have reasonably obvious meanings, but it's not clear what the numbers mean. So let's find out more about the `free` command. Type in

`man free`

and press enter (in future I will assume you know to press enter at the end of a command)

This command introduces us to three new things

- The `man` command, which displays manual pages (in other words, help pages)
- The structure of the command line is

  `[command] [space] [argument]`

  BASH uses spaces to break the command line up into **arguments**. Everything up to the first space is the **command** you want to execute. All arguments are passed to the command (so the command knows more precisely what you want to do). So on this command line `man` is the command, and `free` is the argument passed to the `man` command.
- The output of the `man` command is too big to fit on a single screen, and so the `man` command will send it's output to a *pager* – a program that displays output one screen at a time. Use the up and down arrow keys to move through the document (read it while you are doing this!) and press `q` when you have finished.

From the `man` page (this is what the output of the `man` command is referred to), you may be able to see that the default output is that numbers are in units of kilobytes. They are big numbers, so let's display them in gigabytes to make them easier to understand.

```
sys045@artemis:~$ free -g
             total       used       free     shared    buffers     cached
Mem:            31         31          0          0          5         21
-/+ buffers/cache:          5         26
Swap:            9          0          9
sys045@artemis:~$
```

So the server (apparently) has access to 31GB of RAM (of which almost all is in use).

Let's find out how big the hard disk is

```
sys045@artemis:~$ df -h
df: `/var/lib/lightdm/.gvfs': Permission denied
Filesystem      Size  Used Avail Use% Mounted on
/dev/md0p1      481G  205G  253G  45% /
udev             16G   12K   16G   1% /dev
tmpfs           6.3G  800K  6.3G   1% /run
none            5.0M     0  5.0M   0% /run/lock
none             16G  240K   16G   1% /run/shm
/dev/md0p2      5.0T  522G  4.2T  11% /home
sys045@artemis:~$
```

There is a lot of information here, and much of it is very detailed. But the key figure is that the total disk space is 5TB plus 481GB, with about 11% in use. So there is plenty of disk space, but the RAM needs to be kept under review. This is the sort of stuff a system administrator needs to know.

We note that the numbers seem approximate. From our knowledge of the memory chips and hard disk drives that can be purchased, it is a safe assumption that the system has 32GB of RAM and a 500GB disk drive. The fact the system is reporting slightly lower figures is due to the fact that the system will consume some resources in managing its own internal information. We will cover this in the "operating systems" part of the module.

Look at the `man` page for the `df` command to see if you can find out what else the `df` command can tell you.

Clearly we can provide a Windows-style application to tell us these things. But the command line gives us the flexibility to decide how we want the information displayed, to access the information in some very flexible ways, and to write scripts that will take actions depending on the values returned (we'll look at scripts in semester two).

Creating a directory and a text file

You will all be used to using Windows Explorer to create directories and files. Ubuntu has an equivalent graphical program (called "Nautilus") but we can also do these things at the command line.

If you start Nautilus then you can keep track on what you're doing. The easy way to start Nautilus is to click on the "Home Folder" icon on the Launcher. The techie way is to type

```
nautilus &
```

into the terminal. The "`&`" is new here – that tells BASH to run the `nautilus` command in the background (which means we can carry on typing commands rather than having to wait for `nautilus` to finish)

Using the terminal, type

`mkdir workshop1`

The `mkdir` command is the command to make a directory. `workshop1` is the name of the directory to be created. If you look in your nautilus window then you should see that `workshop1` has appeared as a directory in your home directory.

Now we can create a text file. For now, I will talk you through the process of doing it in two ways:

(1) using the "vi" editor

Space here, but nowhere else

Type

`vi workshop1/vi-file.txt`

Here, `vi` is the command and `workshop1/vi-file.txt` is the argument. Within the argument, `workshop1` is the name of a directory and `vi-file.txt` is the name of a file within that directory. The file does not exist, so `vi` will create it when you save the file.

Once you've started `vi`, do the following things

press `i`
type in some text (such as `this is my file, created with vi`)
press `ESC` (top left of your keyboard)
type `:wq` and press enter. ← Note this is colon w q enter

In your nautilus window, you should be able to look at the contents of the "`workshop1`" directory and see that a file called "`vi-file.txt`" has been created inside it.

`vi` is a **very** complicated editor (but **very** powerful) and I shall be wanting you to learn to use it for basic editing. I admit it is difficult, and if you really can't get on with it then there are alternatives. For now, note that if you deviate from the instructions given here it is likely you'll need me to rescue you! I hope you will become more independent in time.

(2) using the "nano" editor

`nano` is simpler than `vi` but less powerful. It's there if you want to use it, if you really can't get on with `vi`.

Type

`nano workshop1/nano-file.txt`

Here, `nano` is the command and `workshop1/nano-file.txt` is the argument. Within the argument, `workshop1` is the name of a directory and `nano-file.txt` is the name of a file within that directory. The file does not exist, so `nano` will create it when you save the file.

Once you've started `nano`, do the following things

type in some text (such as `this is my file, created with nano`)
press `Ctrl-X` to exit
answer `Y` when asked if you want to save
press `enter` to confirm the filename

Now we've created some files, let's see how to copy, move, rename and delete them.

To copy files, we use the `cp` command

```
cp workshop1/vi-file.txt copy-of-vi.txt
```

You will not get any output (the normal Linux way of working is that a command that has worked properly will produce no output, but a command that has failed will produce an error message). But if you look at your nautilus window, you'll see that `copy-of-vi.txt` has appeared in the home directory (not in the "workshop1" directory).

Note here the command line has the structure

```
[command] [space] [first argument] [space] [second argument]
```

where the first argument is the name of the file that already exists and the second is the name of the new file you want to create

If the target file already exists then the `cp` command's default behaviour is to overwrite it. We can add a command-line switch to modify this default behaviour

```
cp -i workshop1/nano-file.txt copy-of-vi.txt
```

When you are asked

```
cp: overwrite `copy-of-vi.txt'?
```

you can type "`y`" or "`n`" followed by enter to indicate yes or no. In our case, the answer would be no, as you may have noticed we deliberately got the filename wrong.

Note the command line now has the structure

```
[command] [space] [first argument] [space] [second argument] [space] [third argument]
```

and rather than counting arguments it is probably better to say that

- BASH uses spaces to break the command line up into a command and a number of arguments. It starts the `cp` command and gives it a list of the arguments.
- the `cp` command goes through the arguments in order. For each argument, if it starts with a "minus sign" then `cp` assumes it is a command line switch, and uses it to change its behaviour. If it does not start with a "minus sign" then `cp` assumes it is a filename. The first filename it finds is the name of the file that already exists and the second filename it finds is the name of the new file to be created. Once it has gone through all the arguments, it will either make the copy or return an error message.

  Can anyone spot a problem with this?

In a future workshop, we will see how to make the "`-i`" switch be the default behaviour (which will almost certainly save you from inadvertently overwriting the wrong file on many occasions!)

As I said, if you look at your nautilus window, you'll see that `copy-of-vi.txt` has appeared in the home directory, not in the "workshop1" directory. It appear there because that is what you told the `cp` command to do. So let's move it (`mv` is the move command).

```
mv copy-of-vi.txt workshop1
```

You could have done it on one go

```
cp workshop1/nano-file.txt workshop1/copy-of-nano.txt
```

`mv` can also be used to rename things

```
mv workshop1/copy-of-vi.txt workshop1/renamed-file.txt
```

The difference between the two "mv" commands is that, for the first command, the final argument is the name of a pre-existing directory (so it moves the file into the directory). Whereas in the second command the final argument is not a pre-existing directory, so it renames the file.

Now we can delete it (`rm` is the remove command).

```
rm -i workshop1/renamed-file.txt
```

and type "`y`" and press enter to confirm deleting (this is because you used "`-i`")

Because all the files we are using are inside the `workshop1` directory, we are having to type that directory name in in a lot. It's more convenient to tell BASH that we are working inside that directory

```
cd workshop1
```

To see where BASH thinks we are working

```
pwd
```

Whenever we use a filename or directory name, BASH will assume it is inside this "current working directory"

So now we can do

```
cp -i vi-file.txt copy-of-vi.txt
mv copy-of-vi.txt renamed-file.txt
rm renamed-file.txt
```

which does the same as before, but with less typing (and therefore less chance of typing errors).

The final thing I'd like to cover today is to ask how did we know that "`df -h`" was the command we needed to use to. I wanted some information on disk space, so I want to find which commands are something to do with disks. So I use the command

```
apropos disk
```

to get a list of all commands that are something to do with disks. There are too many of them, so I do

```
apropos disk | less
```

to display them using a pager (the `man` command does this automatically, but here I had to do it manually. The | character is to the left of the z key on most keyboards and is used to say that the output of the `apropos` command should be processed by the `less` command – this is known as a pipeline, and the ability to combine commands in this way is something that makes a command line very powerful (we will meet these again)

I can now look at the output of the `apropos` command to see which commands look like they might be the one I want, and then I can press "`q`" to return to BASH, and then look at the `man` pages for any commands I might want to use to see in more detail what they do. In this way, I can find out about commands that I don't know. I can learn new things!!!

In any time left over, note some of the other commands listed by your `apropos` command and look at their `man` pages.

When you've finished, choose "Shut Down" from the "power cog" menu (normally I'll ask you to "Log Out", but for today I want the people in the next workshop to start from scratch).