# Getting started with Linux for G51SYS and G51CSF

Steven R. Bagley

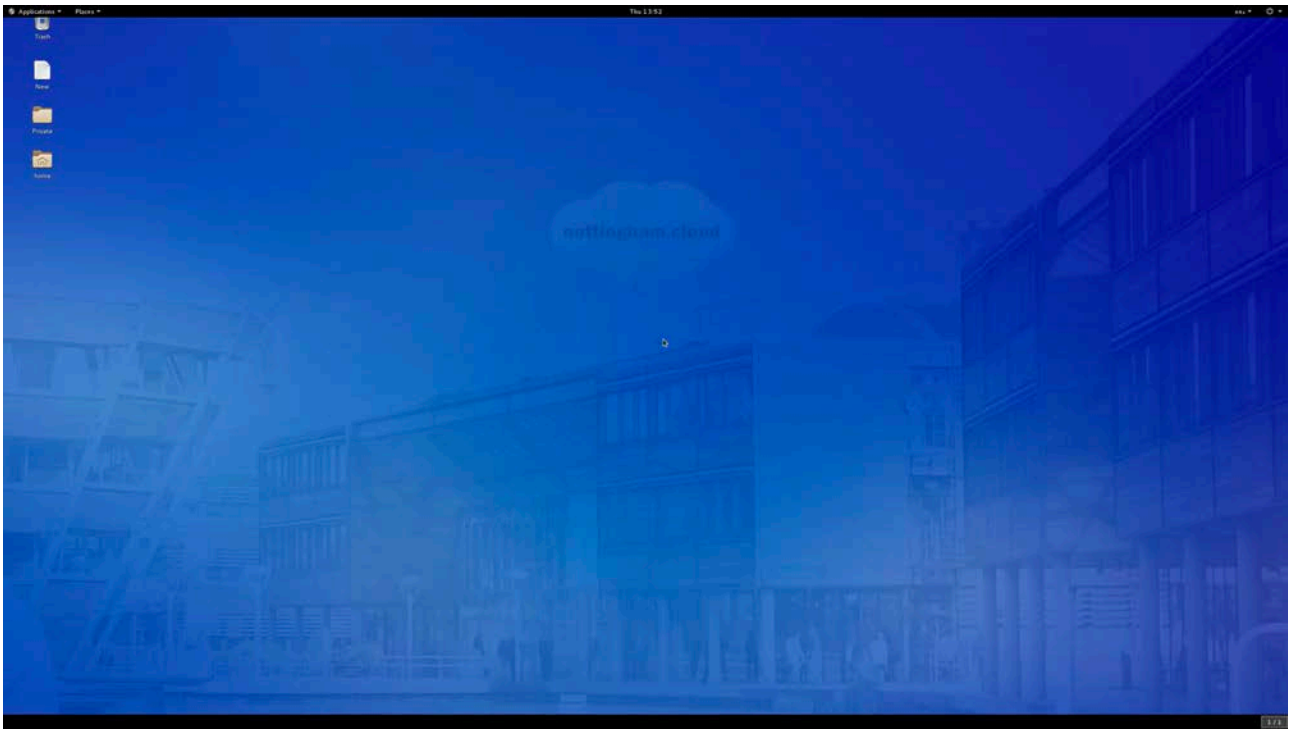
## Introduction

These exercises are very simple and are primarily to get you used to the Linux system, we shall be using during G51SYS and G51CSF. Firstly, we'll see how to login and to move around Linux from the command line, then we'll compile a very simple C Hello World program. We'll download and test the nand2tetris material before finally running a simple ARM program in Komodo.

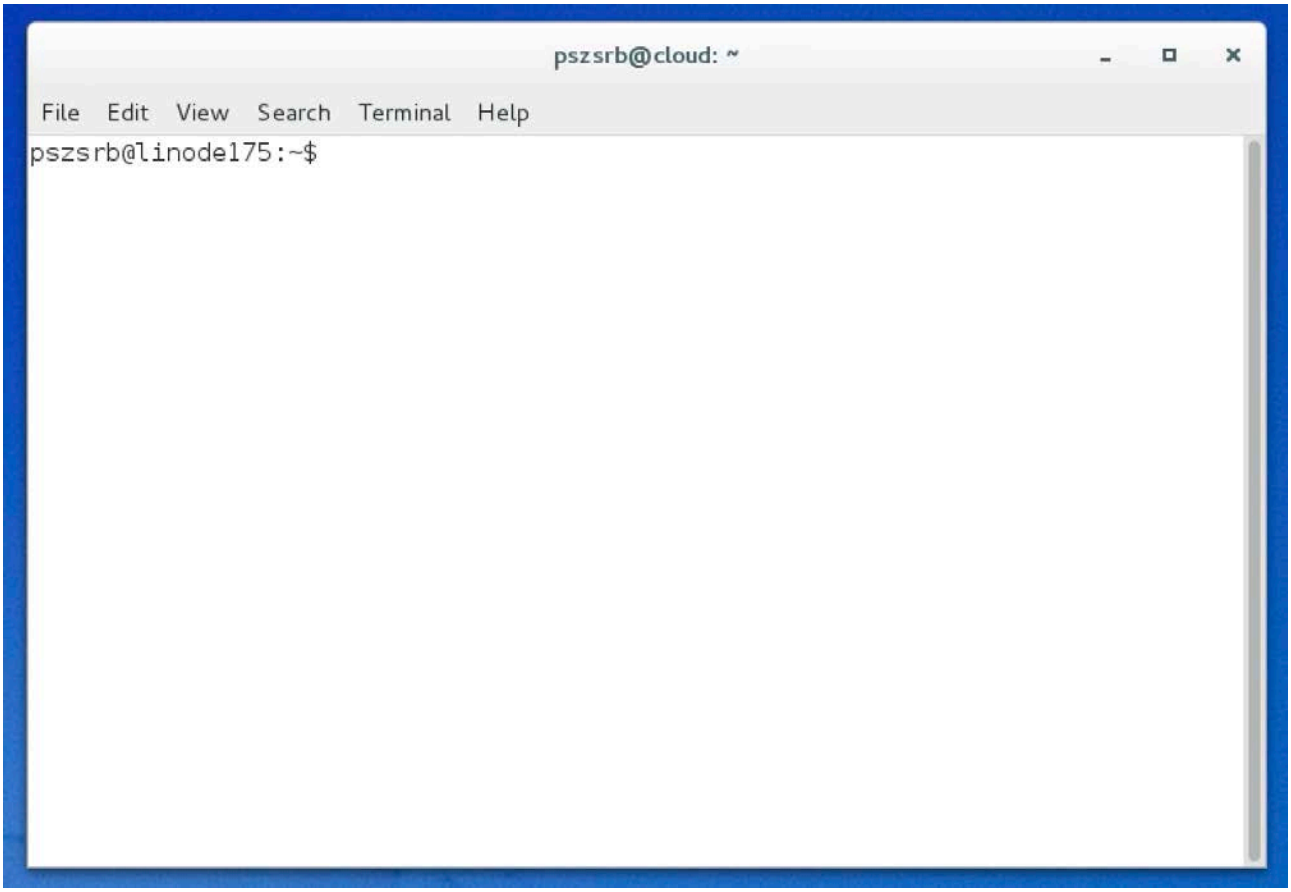
## Logging on

Our Linux system is accessed using the VMWare Horizon Client, this is installed on the fixed desktop machines in A32, but you'll need to download and install it on your own laptop. The easiest way to download it is to head to **http://nottingham.cloud/** and follow the link to download. While you can access it through your web browser, performance will be much better using the native client so it is worth taking the time to install it.

Upon first use, you might be asked to provide the server name to connect to, use **nottingham.cloud**, along with your standard University username and password. Once connected you need to select to the 'Linux Desktop' and connect to it (you may be asked for your username and password again). You should be presented with something similar to this:



Linux is primarily driven from the command line and so we need to open a 'Terminal' that we can enter our commands into. To do this, click on the **Applications** menu in the top-left, and select **Terminal**. You should see be presented with a 'plain terminal' window similar to the one below. We can create more if we need them by following the same procedure. You can now drag and/or resize these windows on the desktop as you wish.




This terminal window allows us access to the Linux command line interface (CLI), which we can type commands into it to make the computer do things. As an example, if you type:

```
ls
```

into the window, it will give you a list of the files in the current directory. The `cd` command is used to change into a directory. If you type:

```
cd Documents
```

This moves into your `Documents` directory. If you type `ls` again, you will see any files or directories currently present. The first thing we are going to do is create a directory (or folder, as it would be known on a Mac or Windows machine) to store our work. The mkdir command creates a directory, and the first parameter is the name of the new directory (note that parameters are separated by spaces so it's generally not a good to use spaces in filenames since it makes things complicated). Let's create a directory to store our G51CSF work, typing:

```
mkdir G51CSF
```

This creates a new directory called `G51CSF` which we'll use for all our G51CSF exercises. If you type `ls` again, you should find your newly created directory listed.

If you change into this directory:

```
cd G51CSF
```

Then `ls` will again show an empty directory. We can move back up the directories by entering the command:

```
cd ..
```

The directory `..` is a special directory that always means the parent directory (there is also a special directory called `.` which means the current directory). Making sure you are in the `Documents` directory, (you can use the `pwd` command to find out which directory you are currently in), create a directory called `G51SYS` to store work for that module (you should be able to work out the syntax of the command).

## Deciding on which UNIX/Linux editor to use

You will need to decide on which text editor you want to use for preparing files on the Linux system. There are several options available and most people develop their own preference. For people new to Linux, **Visual Studio Code**[1] is probably a good bet because it is very similar to text editors you'll have used before. However, feel free to investigate other options such as `pico` (some Linux distributions use an open-source clone called `nano` – see later), `vi`, `gedit`, or `emacs`, but be warned their interfaces can be a little arcane…

## Editing text files

Make sure you are in your `G51CSF` directory (use the command `pwd` to find out where you are, and `cd` to move to the right place — you don't have to issue several cd commands, you can combine them together using a `/` to join path elements so `cd ../G51CSF` means change into the `G51CSF` directory inside the parent directory) and you can now type:

```
code hello.c
```

This will open the Visual Studio Code (you can use a different one if you like and know how it works) to edit the file `hello.c`. If the file doesn't exist, you'll be presented with a blank window, otherwise it will show the contents of the file for editing. VS Code works in a standard fashion for editing files. Enter the classic 'Hello World' program as shown below into the text file. Once you have edited your file, you can save it and go back to the command line.

```
#include <stdio.h>
int main (int argc, char *argv[])
{
      printf("Hello World\n");
}
```

---

[1] If you want to install Visual Studio Code on your own Mac, Windows or Linux setup, you can download it free of charge from `http://code.visualstudio.com/`

## Compiling your file

The GNU C compiler is a UNIX command line tool that is accessed by the `gcc` command just as you have used under `cygwin` on Windows, like this:

```
gcc hello.c
```

This will compile the file `hello.c` into a UNIX executable (i.e. a program) called `a.out` (this is a UNIX compiler convention). This isn't very convenient so we tend to use the -o flag to specify the output filename.

To compile your file just quit your editor and then type, on the command line:

```
gcc -o hello hello.c
```

Here we use the -o flag to name the program `hello`. If all is well no errors will show up, and the command prompt appears once more waiting for the next command. Of course, if there are errors, then you will have to use your favourite text editor to alter your file, followed by recompiling it all over again with `gcc` .

When there are no further errors just type at the command prompt:

```
./hello
```

or

```
./a.out
```

if you didn't explicitly name your binary file. This will cause your program to execute and print out “Hello World”.

## Getting started with the nand2tetris tools

We are going to use the suite of software provided by the **nand2tetris** people for our coursework in G51CSF, so we need to download and extract them onto Linux. Using a web browser (you’ll find several listed under the **Internet** section of the **Applications** menu) such as Google Chrome, navigate to **http://www.nand2tetris.org**, go to the **Software** section and download the software tools using the link at the top.

Once this has downloaded, you will need to copy the `.ZIP` file into the `G51CSF` directory. By default, Google Chrome will have placed this in the Downloads directory inside your home directory. Either copy this into your G51CSF directory using the GUI (double-click the Home icon on your Desktop, and it should work as per Windows) or use the command line to copy it. It is probably worth getting familiar with the command line tools (there’s a crib sheet at the back of this document) so we would copy it using:

```
cp ~/Downloads/nand2tetris.zip .
```

This tells the cp command to copy the file at `~/Downloads/nand2tetris.zip` into the current directory (represented by `.` — don’t forget to type this). Note we are using the special character ~ as a shortcut to refer to your home directory (which everything else is sub-directory of).

Check the `.ZIP` file has appeared in the `G51CSF` directory by using ls to see if the file is now present. If it isn't, check that you copied it into the right directory (if using the GUI), or typed the name of the file correctly.

An alternative method is to use a web browser on the Computer Science Windows Desktop (either, the flexible desktop or fixed machines). On these machines, you should find your Linux file system mounted as the `H:\` drive, enabling you to copy files back and forth between the two systems. Have a look and you should see both the `Documents` directory and the `G51CSF` and `G51SYS` directories we created earlier.

## Extracting the nand2tetris tools

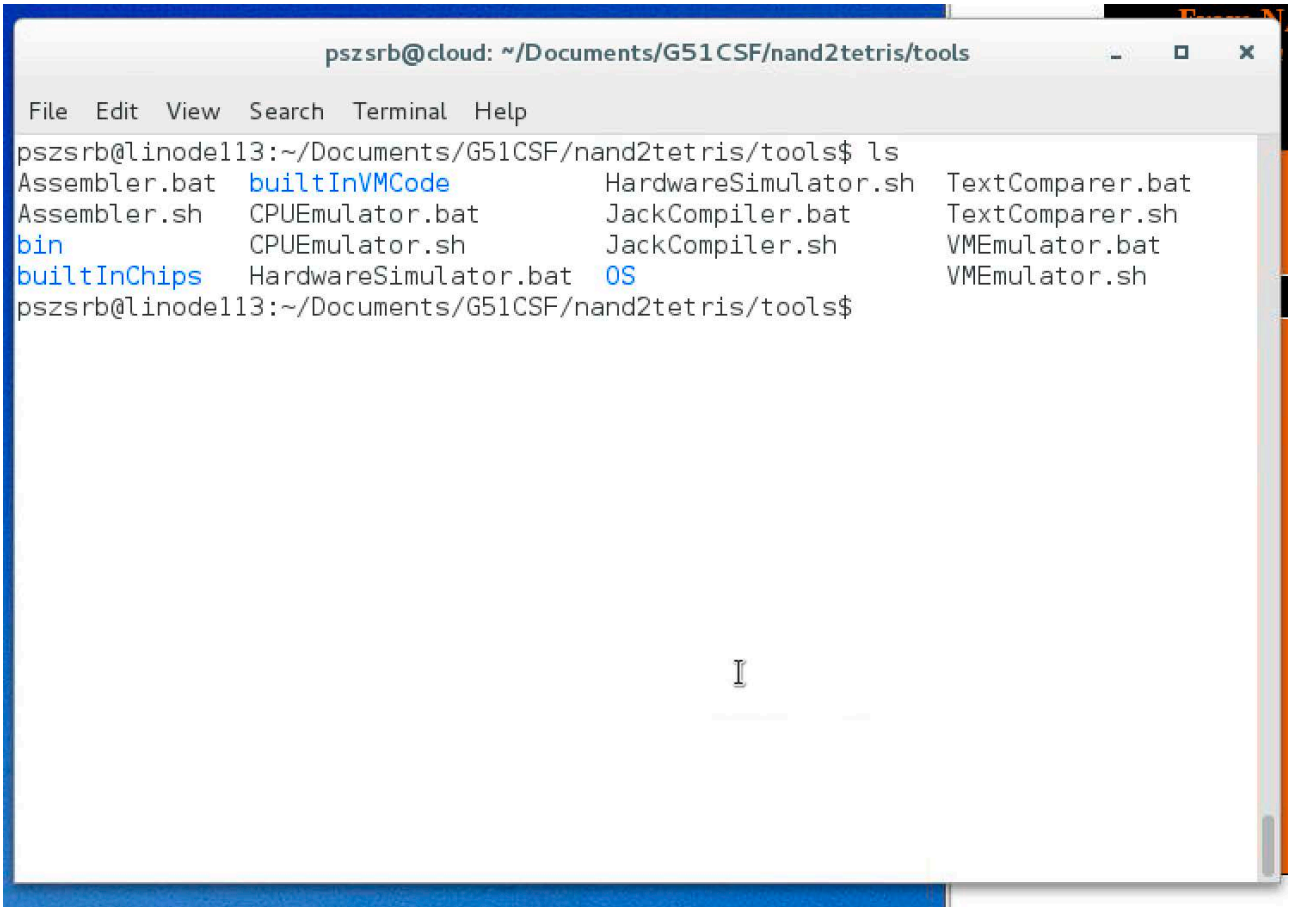
From within the G51CSF directory, type

```
unzip nand2tetris.zip
```

to extract the contents of the file you download. Once this has done, if you list (using `ls`) the contents of the directory, you should see a new directory has been created called `nand2tetris`. This contains all the tools and support files for the module (you can explore this at your leisure using `cd` and `ls` to poke about). For now, we are just going to check the tools work, so we'll jump directly into the tools directory with

```
cd nand2tetris/tools
```

Type `ls` and you should see a list of files similar to these:



Lets check things are working by running the Hardware Simulator, the command is called `HardwareSimulator.sh` and is in the current directory, so type:

```
./HardwareSimulator.sh
```

Unfortunately, you'll probably find you are told 'Permissions Denied', this is because Linux doesn't know this file is executable, so we need to tell it so (note, we only need to do this once). This is done by using `chmod`. While we are doing this, we might as well make sure all the other tools are set to executable, using:

```
chmod +x *.sh
```

The `*` is a special type of symbol called a wildcard on the Linux command line, and means any characters. Here we are using to mean 'any filenames that end `.sh`'. Having typed the above command in, if you type

```
./HardwareSimulator.sh
```

again, you should find that after a moment, the Hardware Simulator starts up. Finally, you should note that we cannot use our Terminal again until we close the Hardware Simulator. We can change this by telling Linux to run the program in the background by putting an & after the command, e.g.:

```
./HardwareSimulator.sh &
```

This works for any program but is generally only useful for GUI ones, such as this.

## Getting started with Komdo (kmd and aasm)

In the window where you have been working, type:

```
cd ..
```

at the command prompt. This returns you to the directory above G51CSF i.e. your `Documents` directory, you may see the directory you are currently in listed in the prompt that Terminal uses, if not use `pwd` to check you are in the right place. We now need to change into the directory for G51SYS we created earlier, i.e:

```
cd G51SYS
```

We are going to run a very simple ARM machine code program that prints Hello World just to check everything is working. As we did for the **nand2tetris** material, download and copy the following file:

```
www.eprg.org/G51SYS/Sources/hello.s
```

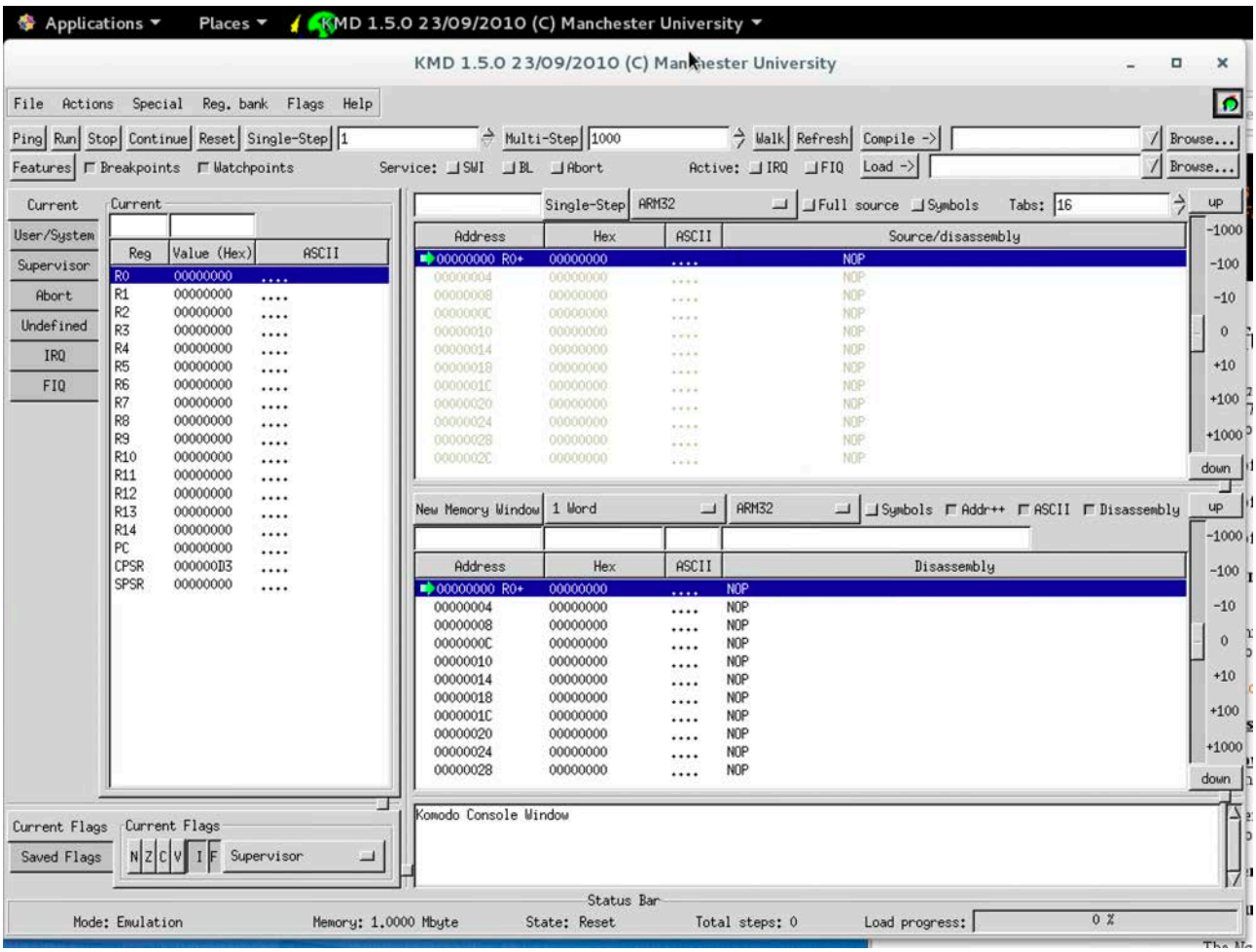
and copy it into your G51SYS directory. Open it with **VS Code** as before and you should find the following program. Don't worry about what it does, we'll cover that in the G51SYS lectures.

```
            B main

hello       DEFB  "Hello World\n",0
goodbye     DEFB  "Goodbye Universe\n",0

            ALIGN

main        ADR   R0, hello  ; put address of hello string in R0
            SWI   3          ; print it out
            ADR   R0, goodbye      ; put address of goodbye string in R0
            SWI   3
            SWI   2          ; stop
```

In the terminal, type:

```
kmd -e
```

to start Komodo. Komodo presents an interactive interface for emulating an ARM CPU — we will look at this in more detail in lectures later but, for now, we are just going to assemble the file we have just download. You should see the following window appear.



Press the **Browse** button next to the **Compile** text box in the top right. Since you are already in the `G51SYS` directory the display should show up your `hello.s` file in the file selector. Select the file and click OK and its full path name appears in the **Compile** text box. Press the **Compile** button and Komodo will assemble the source code for you by automatically calling the `aasm` assembler on the named file.

If all is well there will be no error messages in the monitor window pane at the bottom of the display. Rather it will just tell you that two passes of the assembler have been performed successfully. The program that has now been produced for you is called `hello.kmd` denoting that it's intended only for the Komodo emulator.

Notice that the full path name of `hello.kmd` has appeared in the **Load** text box. So now just press the **Load** button and you'll see an array of binary and source code appear in the various Komodo panes. Don't panic!! Your program is loaded but is not yet running.

Since the program prints out 'Hello World', we need a simulated Input-Output window to see this happen. This is provided in Komodo by just pressing the **Features** button on the left of the display. Now press the **Run** button at the top of the screen and 'Hello World' should appear in your 'Features' window. If you are so bedazzled by this that you want to run it again then press the **Reset** button followed by **Run**.

Finally, press **Reset** one more time followed by successive presses of the **Single Step** button — this executes a single instruction each time it is pressed. Watch the highlighting in the top pane illuminate each instruction, in turn, until eventually, 'Hello World' appears once again. You'll also notice that the values in the left-hand side of the window change. As we'll find out later, single-stepping, breakpoints and watchpoints are vital tools for debugging assembler programs.

# Unix Crib Sheet

## Moving around the file system

| Command | Description | Usage |
|---|---|---|
| `cd` | Change to your home directory | `cd` |
| `cd <dir>` | Change to the specified directory | `cd /cs/ug/fred` |
| `cd ..` | Change to the directory one level above the current directory | |
| `cd ../..` | Change to the directory two levels above the current directory | |
| `pwd` | Print Working Directory (show the current location in the file system) | |

## Listing directory and file contents

| Command | Description | Usage |
|---|---|---|
| `ls` | List the contents of the current directory | |
| `ls -l` | Long listing — will show contents of current directory including permissions, etc. | |
| `ls -a` | List the contents of current directory including dot files (hidden files) | |
| `ls <dir>` | List the contents of the specified directory | `ls /cs/ug/fred` |
| `ls \| more` | Show the directory contents in paged output | |

## To access the manual pages

| Command | Description | Usage |
|---|---|---|
| `man <command>` | Access the manual page for the specified command | `man ls`<br>`man cd`<br>`man man` |

## Listing directory and file contents

| Command | Description | Usage |
|---|---|---|
| `more <filename>` | View the contents of a *text* file in paged output | `more myfile.c`<br>`more ~/fred/A.txt`<br>`more ~/fred/log/log` |
| `cp <file1> <file2>` | Copy `file1` to `file2` (`file1` will still exist, `file2` will be overwritten) | `cp myFile.c newFile.c` |
| `mv <file1> <file2>` | Move `file1` to `file2` (`file1` will no longer exist, `file2` will be overwritten) | `mv activity.log old_activity.log` |
| `rm <filename` | Remove (i.e. delete) a file. Be careful! Most of the time UNIX/ Linux won't ask if you are sure! | `ls /cs/ug/fred` |

## Things to note:

There is a space between a command and its arguments and flags, e.g.

`ls –a myfile.txt` is typed as: `ls <space> -a <space> myfile.txt`
`more myfile.txt` is typed as: `more <space> myfile.txt`

If you are trying to perform an operation on a file which has a space in its name you have to put quotes around the file name. Otherwise, Linux will interpret the spaces in the filename incorrectly, e.g.

```
cp End of Year Report Report2012
```

would need to be typed as:

```
cp <space> "End of Year Report" <space> Report2012
```

UNIX does not have a recycle bin! If you accidentally delete a file you can't get it back without restoring it from a backup.

UNIX will try and do the command you request. If successful and there is no output expected, UNIX won't tell you it has done it, you will just get the command line prompt back. If unsuccessful you will get an error message (sometimes terse).

UNIX commands are case-sensitive.