

# Working on the command line

This sheet will quickly go through a few demonstrations of how to work with a command-line interface. These instructions can be done from a **qinteract** session.

## 1. Moving between directories

The first command is the **cd** command, which stands for “change directory.” Typing:  
**cd**

will change your directory to your “home” directory. This directory belongs to you, and is the Linux equivalent of your `c:\Users\` directory on Windows.

### **cd experiments**

will change directories into the experiments folder. The names of directories and files are case-sensitive.

At any time, you can type **pwd** (print working directory) to see the full path of the directory you are located in. On Linux, paths are separated by forward-slashes. The top-level directory is simply “/”

There are a few shortcuts that can be used with directories-- `~` is an abbreviation for your home directory, so you could type:

<b>cd ~</b>	same thing as <b>cd</b>
<b>cd ~/experiments</b>	goes to the experiments folder in your home directory

to go to your home or experiments folder. You can also use “..” to indicate the directory above the one you are in. Let’s go back to our home folder and type:

### **ls**

This will give a listing of the contents of a directory. Try the command again with the “-l” flag:

<b>ls -l</b>	
<b>ls -l ../</b>	lists the contents of the directory above your home directory

You can make a directory with the **mkdir** command, and remove one (if it is empty) with **rmdir**. Two other useful commands are **cp** and **rm** for copying and deleting files or directories.

<b>cp src dst</b>	(won’t work right now--we have no files to copy yet)
-------------------	--

will copy the file `src` to `dst`. If `dst` is a folder, the copied file will be in that folder with the same name. If `dst` does not exist, then `dst` will be used as the name of the copied file.

Each of the commands that we have used so far is actually a separate program, and furthermore, as a program, each one is a file located somewhere in the filesystem. You can use the **which** command to see where a program is located:

### **which ls**

shows us (on linux) that the “ls” program is located in /usr/bin directory. You can use this to explore some of the different programs that are available:

```
which fsl  
cd /usr/local/packages/fsl-4.1.8/bin  
ls
```

will give you a complete listing of all of the fsl command line programs. Try the same thing to find some of the Biac programs, like **bxhabsorb**.

You can find documentation for almost any command on a Linux system with the man command:

```
man ls  
man fsl
```

If “man” doesn’t give any documentation, sometimes you can find simple usage notes by typing just the program name without arguments:

```
bxhabsorb
```

## 2. File shortcuts

On the cluster, we found fsl in the **/usr/local/packages/fsl-4.1.8/bin** folder.

If you type in:

```
cd /u[Tab]lo[Tab]p[Tab]fsl-4.1.8/b[Tab]
```

You can greatly speed up typing long paths. The **[Tab]** trick can autocomplete your paths by matching what you have already typed against directories that exist.

Another feature that works in a somewhat similar way is pattern matching. You can use **\*** to stand in for any string. So:

```
ls /usr/bin/* lists the many programs that are in /usr/bin  
ls /usr/bin/c* will list only those programs beginning with the letter c  
ls /usr/bin/m*r will list programs beginning with c and ending with r
```

Just like the **[Tab]** trick, the **\*** will only match against files and folders that exist.

## 3. Piping and i/o redirection

Many command line programs print out information to the terminal, like ls does. They may also take input, like the rm command does, if it asks you for y/n confirmation for deleting a file. You can capture output to a file if you like.

```
cd  
ls /usr/bin > file_list.txt
```

First, we went to your home directory. Notice that the next line did not print anything to the screen. If you look in the current directory, you will see a new file called **file\_list.txt** that has just been created. The output from the `ls` command that is normally printed to the screen was instead redirected into this file with the redirect **>**. If it helps you to remember it this way, the **>** looks like a funnel collecting the output of **ls /usr/bin** and funneling it into the **file\_list.txt** file.

We can view the contents of the file with the **more** command. Or we can use the much simpler **cat** command to dump the contents of the file onto the screen. Notice that if you `cat file_list.txt` the file is dumped faster than you can read it.

Another useful command is the **grep** command, which lets you search a file for a pattern.

### **grep -n cd file\_list.txt**

Will search for lines containing “cd” inside the file **file\_list.txt** and number them.

So, what we have done with the last couple of commands is to use `ls` to print a listing of files, redirected it into a text file, then searched that text file for a pattern. There is a shorter way to accomplish that using the “|” command. This is called a “pipe.” It allows the output from one program to be redirected as input into another program. Try

### **ls /usr/bin | grep cd**

Or try first:

**ps -eF** which should give a list of every program running on the node, and then **ps -eF | grep yourlogin** to see only your own programs.

The output from the first command was printed to the screen as a table of all running programs. For the second command, the table was generated, then sent to the **grep** program, which searched it, and only printed the lines that contain **yourlogin**. This may seem pretty complicated, but a large part of the power that comes from using the Linux command line is the ability to string different programs together to accomplish more complicated tasks. Many of the Linux commands are designed to be used this way—to print output in a way that it can be used by other programs, or to be able to take in and process the output of another program.

Only one program at a time has the ability to print to the terminal, or take input from the keyboard. This is why we use the **&** when we run **gedit**. Running without the **&**, **gedit** will receive keyboard input from the terminal, and can print output to the terminal. But, it doesn’t need either of these things—it is a graphical program. Running **gedit &** forces the program to relinquish its control of the terminal.

## 4. Setting environment variables

One of the final things to cover on the command line is the setting of environment variables. These are variables that you can set and retrieve, just like you can with any other programming language. But they are also inherited by any other program that you run on the command line. They can be used to configure or change the way that programs run. There are already quite a few environment variables set on your behalf, and you can see them by typing

**printenv**

You can see that many of these contain configuration information for FSL—like telling FSL where it is installed.

To make a variable, you can just type something like the following:

```
MYVAR="hello"
```

Notice that there are no spaces, and the value of the variable is in quotes. Variables are usually written in all caps as a convention. We can use the variable by prefixing it with a **\$**

```
ls $FSL_DIR/bin
```

```
ls $FSL_DIR/bin > $MYVAR.txt
```

To check how the variable replacement has worked, you can use the echo command. This simply repeats back the input, after the substitutions have happened.

```
echo $FSL_DIR
```

```
cd
```

```
echo ls $FSL_DIR/bin > $MYVAR.txt
```

This is very useful for debugging scripts. So far, we have set variables and used them. To set an environment variable, use the export command.

```
export MYVAR
```

This will make the variable visible to other programs. Also, you should now be able to see it when you type

```
printenv
```

This is probably quite a lot to take in all at once, and a bit confusing, but hopefully this worksheet can be an in-hand reference to many of the commands that you might use when putting together a script, or working on the command line. It covers many of the basics, but also includes a few tips about how to start to explore the system (using **which** to find directories with programs, using **man** to find documentation) that will be helpful when you want to start doing more than what is covered today.