Free Electrons

Introduction
to the Unix command line
Training lab book

This training document is no longer being updated by Free Electrons. It may be obsolete.

Introduction to the Unix command line
Training lab book

Free Electrons
http://free-electrons.com

100%
RECYCLED PAPER

Free Electrons

Introduction to the
Unix command line
Training lab book

About this document

This document is part of an embedded Linux training from Free Electrons.

You will find the whole training materials (slides and lab book)
on http://free-electrons.com/docs/command-line/

Copying this document

Document updates and translations available on http://free-electrons.com/training/intro_unix_linux

Corrections, suggestions, contributions and translations are welcome!

Training setup

Instructions for the instructor, or for self training people. Otherwise, you can proceed to the next page!

The training session can be performed on any GNU / Linux system, in particular on a live cdrom like Knoppix, Fedora Core and Ubuntu.

Download the lab files directory from
http://free-electrons.com/labs/unix-linux-introduction-labs.tar.xz and extract the archives in a convenient directory. In the labs, we will assume that this directory is /mnt/labs.

Disclaimer

Do not take the contents of the training lab files for granted!

Sardars are very nice, friendly and even clever people originating from India. I know some of them in person, and I can certify this! This is probably why other people in India are not afraid of telling kind jokes about them.

Microsoft is a perfectly honorable and trustworthy corporation, which only pursues the best interest of individuals and society as a whole, in a truly altruistic way.

Introduction to the
Unix command line
Training lab book

Free Electrons

Lab 1 – Basic file handling

Objective: getting familiar with basic file handling.

At the end of this lab, you will be able to

- Display the list of files in a directory, and see their contents.
- Perform simple operations: copying, renaming, removing, creating links.
- Modify access rights to files and directories
- Et even get rid of Microsoft!

Setup

Go to the `/mnt/labs/intro/files` directory.

List of files

Display the list of files. How many files and directories are there?

Which is the hidden file?

List the oldest files first, and the most recent last.

Now, list the smallest files first, and the biggest last.

You will find the option that you need in your course, or by consulting the `ls` command manual page.

Accessing file contents

Display the contents of the `answering-machine.txt` file at once.

Now, display them with a tool which stops at the end of each page and waits until you hit a key, to leave you time to read.

After reading a few pages, directly go to the part of the text containing the `planet` word. If you use one of the 2 most popular tools (suggested in the lecture), you can do this by using the "/" command (inside the tool). Then, you can go to the next occurrence of this word with the "n" command.

Once you reach the end of the file, look for the last occurrence of the "BEEP" word, then continue upwards to the previous BEEP words, in the same way.

Note that only the best of the 2 tools highlights the search word in reverse video. If you don't observe this, it means you are not using the best one!

Display the last 20 lines of the `sardar3.txt` file.

Enter your command without typing the whole file name. Just type the beginning of it, and hit the Tab key.

Note that you can find the same commands in the `vi` editor too. Though standard Unix commands are independent, there is some consistency between them!

Searching through file contents

Look for `trust` in `Microsoft`.

Look for `money` in all the files in the directory (included in the subdirectory).

Without typing again the full command, now look for `Money`.

Again without typing again the command, look for the same word, whatever its case.

Do it by editing the previous command.

Free Electrons

Introduction to the
Unix command line
Training lab book

Making changes on file and directory names

Modify the name of the `.lightbulb` file so that it is no longer hidden.

Get into the `sardar/` directory. Check that you are in the right directory. Move the `sardar3.txt` file from the parent directory to the current directory.

Go back to the parent directory.

Get rid of `Microsoft` once and for all.

Create an `archives` directory and copy all the files in the working directory into it, including the `sardar` subdirectory and all the files it contains.

To check the name of the current directory, the `pwd` ("print working directory") command is your friend.

Symbolic links

Create symbolic links making the files in the `sardar/` directory appear in the current one too.

Once more, list the files in the current directory. Are links easy to identify?

Remove the `sardar/sardar3.txt` file and see the impact on the file list.

File access rights

Try to suppress the `sardar/sardar1.txt` file.

Display the access rights of the various files and try to understand why you are not allowed to do it.

Once you understood, change these rights and remove this file.

Now, try to get into the `safe` directory. Modify access rights to be able to do it.

Once you're inside, your adventure is not over yet. As you are only interested in `fortune`, remove the `-o` file which is also there. Good luck!

As these practical labs do not involve several users at the same time, it is not really possible to propose elaborate exercises on file access rights. You will have your own practice on real-life GNU/Linux servers!

Free Electrons

Introduction to the
Unix command line
Training lab book

Lab 2 – Elaborate commands

Objective: get familiar with redirections, pipes and task
control

At the end of these practical labs, you will be able to

- Redirect the output of a command to a file
- Implement pretty complex requests by cascading multiple Unix
  commands.

Setup

Go to the `/mnt/labs/intro/commands` directory.

A first redirection

Use the `history` command to show all the commands that you
already typed.

Now, save the output of this command in a new `history.txt` file.

It is sometimes useful to keep track of typed commands. The `history` command is definitely useful in this case.

Concatenating files

Concatenate all the files in the `sardar/` directory into the
`sardar_power.txt` file, still without leaving the current directory.

How many lines, words and characters are there in this new file?

Display all the lines in this file containing the `singh` keyword (case
insensitive)

Remove the `sardar_power.txt` file.

Using pipes

In only one command line, display again all the lines in the files in
the `sardar/` directory which contain the `singh` keyword (case
insensitive).

We will get the same results, but without using an intermediate file. This shows the power of pipes!

Now, count the number of lines this represents, still in a single
command line.

Modify this command to only count the lines containing both `santa`
and `singh`, still in a case insensitive way.

Improve once more the command to count only lines containing
`santa` and `singh`, but not `banta`.

You have just discovered all the power of Unix pipes! These pipes let
you do exactly what you need to, from very simple basic commands.

That's all for the moment! Go on practicing commands introduced in
the lectures, on files available your GNU / Linux system.

Introduction to the
Unix command line
Training lab book

Free Electrons

Lab 3 – Text editors

Objective: get more familiar with text editors

At the end of these practical labs, you will be able to

- Move a whole block of text, creating or removing a margin easily.
- Edit text in a text only console, through the `vi` editor.

Setup

Go to the `/mnt/labs/intro/text` directory.

nedit features at a glance

Install nedit with `sudo apt-get install nedit`.

`nedit` is a user friendly yet powerful text editor.

First open the `msreligion.txt` file with `nedit`.

Review the contents of the various menus, and don't hesitate to try
the corresponding options and commands.

For example, toggle `Incremental Search Line` in the `Preferences`
menu. Use the new dialog to search for all the occurrences of the
`Microsoft` word (whatever its case). Easy, isn't it?

Use the `Return` key to go from one
occurrence to the next.

Now try `Windows -> Split Window` (no hidden meaning this time).
You can work on 2 parts of the text at the same time. Get back to the
single pane mode.

Convert the whole first paragraph into uppercase.

Now, a very nice feature: holding down the `[Ctrl]` key, select the
contents of the first paragraph, but not the spaces on the left and
right hand sides.

Now press the middle mouse button (or both buttons if you don't
have any), and move your mouse! You see you can easily remove the
space margin on the left of the selection. You can even move the
selection over another part of the text and cover it!

Unlike traditional Unix text editors,
`nedit` implements most of the now
universal keyboard shortcuts:
`[Ctrl][c]`: Copy
`[Ctrl][v]`: Paste
`[Ctrl][x]`: Cut
`[Ctrl][z]`: Undo
`[Ctrl][a]`: Select all

Another way of achieving the same thing: in the same way, select the
space box on the left of the 5 next paragraphs. Hit `[Ctrl][x]`, and
the empty space is gone!

Editing text with `vi`

Install vim with `sudo apt-get install vim`.

First, make sure you have the `vi` command summary page from the
training slides in front of you!

Open the declarations.h file with `vi`.

Moving the cursor

Note that you can use the arrow keys to move the cursor in the text.
Now, do the same thing without using the arrow keys.

Similarly, move one page forward and backwards without using the
`Page Up` and `Page Down` keys.

It takes time to get familiar with this
way of moving the cursor. However, as
you don't have to move your fingers
away from the center of the keyboard,
it is a very efficient way!

Free Electrons

Introduction to the
Unix command line
Training lab book

Inserting text

Go to the first line of the text.

Enter insertion mode and add `/*` at the beginning of the line. Exit insertion mode when you are done.

Go to the end of the line by using a single command (and not by moving the cursor character by character!)

Insert a space at the end of the line, followed by `*/`. Exit insertion mode again. Just as before with the end of the line, go back to the first character of the line by using a single command.

Insert the below line before the first line:

```
long horn;
```

Exit insertion mode and place the cursor anywhere on the second line. Start editing a new line right after the current line. Invent your own funny C declaration!

Save your changes and exit `vi`.

Finding words

Open the `pastacode.txt` file with `vi`.

Use a command to go to the first occurrence of the `code` word.

Go to the next occurrence by using the `n` command. Once you reach the end of the file, make a search backwards for the `software` word. Similarly, go to one occurrence to the next.

Replacing words

Look for a particular word using the search commands. Modify the entire word by using a single command.

Now, modify the next occurrences of this word by using the "`.`", command, which repeats the latest edition.

Also remove entire words, or just a given character.

Find how to replace a single character by several characters.

Find how to delete one line in a single command. Find how to delete 10 lines, still in a single command.

Now, how would you replace the 5 next characters starting from the cursor?

That's all for today! You can continue to try more `vi` features on your own from your command summary sheet. The instructor can also show you more.

Side notes:

You can now see that there are 2 modes in `vi`: insertion mode and command mode.

Unlike traditional `vi`, `vim` (the `vi` implementation you are using), `vim` highlights all the instances of the search words.

This "`.`" command is definitely one of the most useful commands in vi!

Free Electrons

Introduction to the
Unix command line
Training lab book

Lab 4 – SSH

Objective: getting familiar with SSH

At the end of this lab, you will be able to

- Use SSH for remote connection and remote command execution
- Use keys and agents to simplify the connection process

Setup

Go to the `/mnt/labs/intro/ssh` directory.

Install the SSH server

By default, only the SSH client is installed. Start by installing the
SSH server by installing the `openssh-server` package.

Remote connection

Once the SSH server is installed on your neighbor machine, ask him
to create a new user account for you (using the adduser command).
Then connect to this account from your machine using SSH.

You can run commands as usual, but they are executed on the remote
machine.

Keys and agents

Create your own pair of private/public keys by using the command
`ssh-keygen -t dsa`. Once created, you have a file `~/.ssh/id_dsa`
which contains your private key (should never be sent to anybody, it's
private !) and `~/.ssh/id_dsa.pub` which contains your public key
(which can be made available to others).

Transfer your public key to the remote host using `ssh-copy-id`. You
should now be able to log to the remote host by entering your SSH
passphrase instead of the password.

Make sure that the SSH agent is running by looking at the list of
running processes in the system for a process named `ssh-agent`. By
default, on Ubuntu, the SSH agent should be running.

Tell the SSH agent what your passphrase is by using the `ssh-add`
command. Once the SSH agent knows your passphrase, you should
be able to log in to a remote host which knows your public key
without entering any password. Very useful for scripting!

Free Electrons

Introduction to the
Unix command line
Training lab book

Lab 5 – Application development

Objective: Learn the basics of application development under Linux

At the end of these practical labs, you will be able to

- Use gcc to compile applications
- Use third-party libraries
- Use gdb to debug applications

Setup

The lab informations are present in the `/mnt/labs/intro/appdev` directory.

Compile a simple application

Write a simple Hello World application in a .c file with your favorite text editor. Then, compile this application:

```
gcc -o myapp myapp.c
```

Finally, run your application:

```
./myapp
```

Simple, isn't it ? :-)

Write a Makefile

First, split your applications in two files, with the `main()` function in `file1.c` calling another function in `file2.c`, for testing purposes.

Then, using the training slides and the `make` documentation, write a simple Makefile that allows to compile this application. Make sure that when one source file is modified, only this file gets recompiled.

Don't forget to implement a `clean` target in your Makefile.

Using a third-party library

We will use the Gtk library to develop a simple graphical application. Under Ubuntu, the Gtk library is available in the `libgtk2.0-0` package, and the development files in the `libgtk2.0-dev` package. Install both of these packages. You can also install the `libgtk2.0-doc` package to get the Gtk library documentation.

Gtk supports `pkg-config`, so if you run `pkg-config --list-all` to see all the libraries supporting `pkg-config`, you should see `gtk+-2.0`. Now, run the command "`pkg-config --cflags --libs gtk+-2.0`". It should display the compiler and linker flags needed to compile a Gtk application.

Look at the source code provided in the `gtk-helloworld.c` file in the lab informations directory. It implements a simple Hello World application in Gtk, which displays a window with a button, and an action associated to the button when it gets clicked.

Compile this application:

```
gcc -o gtk-helloworld gtk-helloworld.c \
```

Free Electrons

Introduction to the
Unix command line
Training lab book

```
$(pkg-config --libs --cflags gtk+-2.0)
```

Debugging an application

Compile the crashing.c application:

```
gcc -o crashing crashing.c
```

When you run this application, an error occurs: «Segmentation fault». It means that the application made an invalid memory access. No harm for the system thanks to memory protection, but we still would like our application to work.

Recompile the application with the `-g` gcc option, and run it inside gdb. You will see when the application crashes and why it crashed. Of course, in our example, the bug is very simple, but `gdb` can help you similarly in more complicated situations.

Now, let's discover the mechanism of core files. A core file is generated when an application crashes. This file contains an image of the memory at the time of the crash, which allows later inspection of the memory contents to understand the cause of the crash.

First, to enable the generation of core files, run:

```
ulimit -c unlimited
```

Then, run the application again. Instead of « Segmentation fault », you should see « Segmentation fault (core dumped) », and a file named «core» has been generated. Now, while the application is already terminated, we can explore its state at the moment of the crash using `gdb`:

```
gdb -c core crashing
```

Finally, let's try `gdb` on another application, implemented in `looping.c`. Compile this application (don't forget the `-g` option!) and run it: it loops forever, printing something on the screen.

While the application is running, attach `gdb` to it using:

```
gdb looping -p PID
```
where PID is the process ID of the running looping process, that you can find using `ps` or `top`. Or you can also run:

```
gdb looping -p $(pidof looping)
```

Once attached, gdb automatically stops the application. By running bt you can display the function call stack. The application is probably running somewhere in the C library, inside the `printf()` C library function, called by the showtests() function implemented in our application.

As the printf() function is called at looping.c line 20, set a breakpoint at this location:

```
break looping.c:20
```

Then, run cont to continue the execution of the program. It should always immediately hit the breakpoint and halt at `looping.c` line 20. If you run cont again, it will hit again the same breakpoint.

Once the breakpoint has been reached, you can use the `list` command to display the source code around line 20. The source code contains a loop that loops until `tests[i].a == 0`. You can print the different variables :

```
print i
print tests[i].a
```

Free Electrons

Introduction to the
Unix command line
Training lab book

Because the bug is easy, you'll very quickly understand that we
forget to increment i in the loop!

Free Electrons

Introduction to the
Unix command line
Training lab book

Lab 6 – Version control with CVS

Objective: get familiar with basic CVS features

At the end of these practical labs, you will be able to

- Initialize a repository
- Access a remote repository
- Import an exiting project in a repository
- Create and use a working copy
- Handle conflicts

Setup

Go to the `/mnt/labs/intro/cvs` directory.

Working alone, locally

In your home directory, create a `cvsrepo` directory that will contain the CVS repository, and initialize this repository using the `cvs init` command.

Download the latest version of TSLib from its website <http://tslib.berlios.de/>. TSLib is a library to handle touchscreen panel events. We will use it as a sample project.

Uncompress the TSLib archive, and then import its source code inside a `tslib` module in the CVS repository. Importing source code in a CVS repository is done using the `cvs import` command.

Once TSLib is imported, create a working copy of the tslib module using the `cvs checkout` command. You are now ready to work on TSLib.

Make a simple modification to a source or documentation file, and look at this modification using `cvs diff`. Once you are satisfied with your modification, send it to your repository using `cvs commit`. A new version of the modified files has been created, and you are now sure that you can revert back to the previous version!

Working remotely

Now, agree with your neighbor on a repository location that you will share. For example, you will keep using your local repository, and your neighbor will access the same repository remotely through SSH.

First of all, make sure that your neighbor can login to your machine. Then, create a group named `cvsusers` (using the `addgroup` command) and add yourself and your neighbor inside this group:

```
adduser yourself cvsusers
adduser neighbor cvsusers
```

Finally, change the group owner of the CVS repository to cvsusers, and make sure that the group has writable access to the repository:

```
chgrp -R cvsusers /home/user/cvsrepo
chmod -R g+w /home/user/cvsrepo
```

These operations make sure that your neighbor has write access to the repository. He should now be able to create a working copy of

Free Electrons

Introduction to the
Unix command line
Training lab book

your repository using:
```
cvs -d :ext:user@host:/home/user/cvsrepo/ checkout tslib
```

Once done, make a modification in one of the working copy and commit this modification. From the other working copy, use `cvs update` to get the modification from the repository. Sharing changes made easy!

Now, let's generate a conflict to see how they are resolved. To generate a conflict, follow these steps:

- First, make sure the working copies of yourself and your neighbor are properly updated and that no modifications are pending (`cvs diff` should return nothing);

- On one working copy, make a simple modification in a file and commit this modification;

- On the other working copy, without updating, make another simple modification to the same file around the same location. And commit this modification;

- It should deny the commit, saying that the file is not up-to-date, so run `cvs update` to get the newer version.

- `cvs update` will try to merge the newer version with your modifications, but will fail because the modifications occurred at the same location in the same file, so it warns you that a conflict occurred.

Now that the conflict is generated, you must solve it. Edit the file in conflict and resolve the conflict by removing the conflict markers and choosing the correct version or merging both versions. Once the conflict is resolved, commit the final version.