UNIX TUTORIAL

Welcome to this tutorial. In this document we will learn how to navigate, control, view and use the hidden power of Unix via the Command Line Interface (CLI).

This tutorial is intended as a first Unix contact and explains only the basics of Unix commands. We assume that you are logged into a Unix environment, such as Ubuntu Linux or Mac OS X and that you have started a Terminal application (Unix emulators like Cygwin on Windows are not recommended!).

Whenever you encounter the symbol →, your active work is expected: Enter the command following the symbol '%', and terminate the command with <Return>.

1. Command Line Interpreter (CLI)

Unix users will find it easiest to interact with the CLI using a terminal application or remotely via an SSH compliant telnet client. By accessing the CLI directly, users can execute commands that might not be available via the Graphical User Interface (GUI), or run installation programs or binaries from other UNIX builds, like OpenBSD, FreeBSD or Linux. For Windows users, the CLI is similar to the DOS prompt gained by typing “cmd” at the run command.

The CLI looks rather strange to the GUI-trained eye. There are no buttons, no mouse pointer or visible menu structure. The CLI basically works through a series of text commands and control keys.

One thing to understand about the CLI is that there is very little or no error checking available through this interface. Users must be very careful when using the CLI to communicate to the UNIX kernel.

1.1 Read-Eval-Print Loop

CLI provides a flexible read-eval-print loop. That is, a loop that reads your command, evaluates it, and prints the result. It's that simple. The command line begins always with a prompt, hereafter we use `'%'`; the rest of the line has to be typed by the user.

→ Print today's date:

```
% date
```

1.2 Unix Command Syntax

Unix commands are usually very short, small and simple. Most Unix commands have options, and some of them accept input files. The basic syntax of a command is :

```
<command> [<options>] [<input>]
```

→ Check out the effect of `-u` option for `date` and `-n` for `echo` :

```
% date -u
% echo hello world
% echo -n hello world
```

1.3 Manual Pages

In all Unix shells, the line commands are described in an online manual, called the man pages.

→ Get the manual for `echo` and `man` :

```
% man echo
% man man
```

For Unix newcomers, the manual pages are somewhat cryptic, but as usual in Unix philosophy, their structure is simple : First, the syntax and semantic of the command are stated, then all the possible options are given and explained. Hint: Press `'q'` or `'<ctrl>-z'` to quit man.

2. File Manipulations

In Unix, everything is a file (even the keyboard, the screen, the hard disks and all other devices are treated as files !). Some important commands to handle standard files are `ls` (list), `mv` (move), `cp` (copy), and `rm` (remove).

→ Before we begin our file manipulations, we will create the directory `atwork` in the home directory (`'~/'` is an alias for the home directory), and change the current directory to it :

```
% mkdir ~/atwork
% cd ~/atwork
```

2.1 List, Create and View Files

→ List the files in your current directory :

```
% ls
```

→ Create a simple text file `hello.txt` with the content `hello world` (`'>'` is an indirection operator), and list the files :

```
% echo hello world > hello.txt
% ls
```

→ View the file you have created :

```
% cat hello.txt
```

2.2 Move and Copy Files

The `mv` command moves a file.

→ Rename the file you have just created into `helloworld.txt` :

```
% mv hello.txt helloworld.txt
```

→ Verify this by listing the files again:

```
% ls
```

The `cp` command allows you to copy a file into another file.

→ Create another instance of your little text file:

```
% cp helloworld.txt hello.txt
% ls
```

2.3 Delete Files

In order to delete a file, you remove it with the `rm` command.

→ Remove the file `hello.txt` :

```
% rm hello.txt
% ls
```

3. Directory Manipulations

Directories are special files (a kind of containers) that help you to organize your files.

3.1 Tree Structure

A directory can contain standard files, directories and any other special files, like an external hard disk. This leads to a tree structure, where directories are the nodes and all other files are the leaves. The root of this tree is called the root directory.

There are special signs allowing you to navigate through this tree structure :

`/` : root directory	`.` : current directory
`~` : home directory	`..` : parent directory
`-` : last visited directory	

→ Try out following commands :

```
% ls /
% ls ~
% ls ..
```

3.2 Working Directory

The current directory is called the working directory.

→ Try out the following two commands (and explain why they are equivalent) :

```
% ls
% ls .
```

→ The command `pwd` displays the full pathname of your working directory :

```
% pwd
```

→ You can change the working directory with the `cd` command, for example change to the parent :

```
% cd ..
```

→ and change back to the last visited directory :

```
% cd -
```

3.3 Creating and Removing Directories

Directories are created with the `mkdir` command.

→ Create a subdirectory `test` in your `atwork` directory:

```
% cd ~/atwork
% mkdir test
% ls
```

The command `rmdir` removes a directory, but it works only if the directory is empty (you can remove a non empty directory with the command `'rm -r'`).

→ Try:

```
% rmdir test
% ls
```

4. Input and Output

A program has commonly an input and output (I/O). For C programs, the standard I/O is the terminal, i.e. input is readed from the keyboard and output is printed to the screen. However, input and output can be redirected using the `'<'` (input), `'>'` (output), and `'>>'` (append) redirection operators.

→ When you type :

```
% echo hello > hello.txt
```

you actually redirect the output of echo into a text file called `hello.txt`. And with :

```
% echo hello >> hello.txt
```

you append the output of echo to the end of the file `hello.txt`.

Another useful tool for redirection is the pipe `'|'` which stores the output of a program in a FIFO (First In First Out) buffer and provides its contents input to another program.

→ Test the following command in your home directory (`wc` counts lines, words and characters):

```
% ls -l | wc
```

→ Note that the following three commands produce the same result (the pipe `'|'` also creates a temporarily file, which is removed at the end of the `wc` call !):

```
% ls -l > temp.txt
% wc < temp.txt
% rm temp.txt
```

5. Summary

In this tutorial we have seen following commands:

`date`	Display date
`echo`	Print a string to the output
`man`	Open the manual pages
`mkdir`	Make a directory
`cd`	Change the working directory
`ls`	List a directory
`cat`	Concatenate and print files
`mv`	Move a file
`cp`	Copy a file
`rm`	Remove a file
`pwd`	Print the working directory
`rmdir`	Remove a directory (only if empty)
`p | q`	Pipe the output of *p* to *q*
`p < f`	Redirect the input of *p* to *f*
`p > f`	Redirect the output of *p* to *f*
`p >> f`	Redirect the output of *p* and append it at the end of *f*
`wc`	Count lines, words and characters

Acknowledgement. Many students and assistants have in the past successfully contributed to this tutorial. Oliver Hitz, Amine Tafat, and Fulvio Frapolli have particular merits.