CS 114 – Fall 2004

Lecture 1 – Monday, September 27, 2004

Introduction to Unix

When you log in to a Unix machine (either through `ssh`, `telnet`, or by walking up the machine), you will be presented with a login prompt that looks like:

```
login:
```

The system is asking you for your *username*, the identifier by which you are known to the system. Be default on `turing`, your username is just your Cornell NetID. After entering your username, the system then asks you for your password:

```
login: username
Password:
```

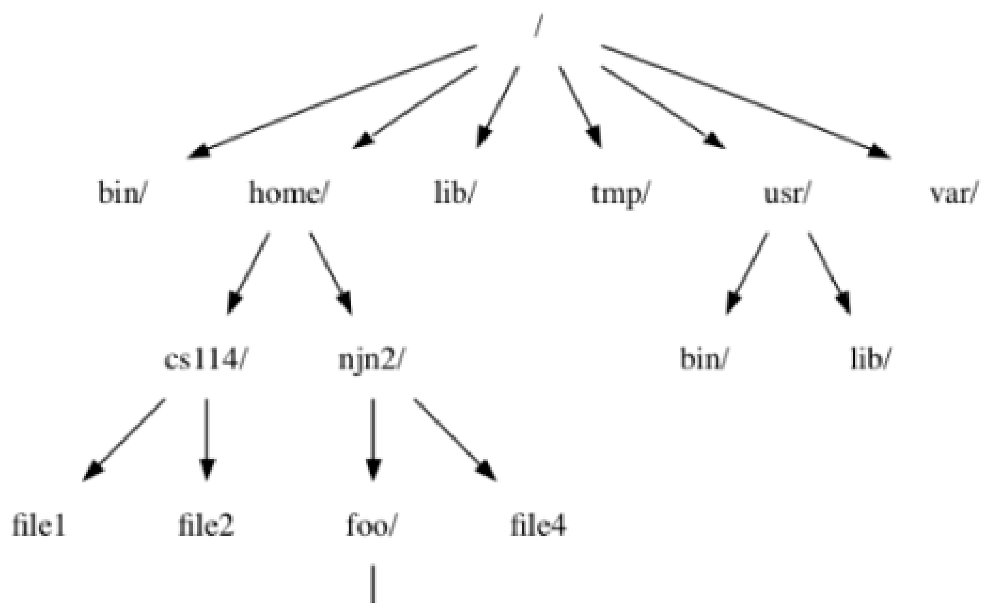
When you type in your password, the console will not echo what you type back to you. This is to prevent others from learning your password by peering over your shoulder. If you enter your username and password successfully, you end up at the Unix prompt, which on `turing` looks like:

```
turing%
```

The prompt is where you interact with Unix. Typically, you issue commands, possibly with arguments. Some commands start programs, others manage the environment, and others give you information.

Everything under Unix lives in *files*: programs, data, etc. These files are distributed amongst different directories. Directories contain files, and may also contain other directories. Files thus form a tree-like hierarchy starting from a base directory called the *root directory*. This hierarchy is often called the *filesystem*. The filesystem may be on a local disk, or it may be across the network; we will not worry about exactly where the files are physically stored. Each user on the system gets a special directory, called their *home directory*, where they can put their files and create subdirectories, etc.

Here's a typical filesystem. Directories are indicated by names with a trailing `/`:



In English, the root directory `/` contains 6 subdirectories `bin/`, `home/`, `lib/`, `tmp/`, `usr/`, and `var/`. Of those, `home/` and `usr/` have content indicated in the graph. If we look at `home/`, it contains two directories `cs114/` and `njn2/`. The directory `cs114/` contains two files, `file1` and `file2`, and `njn2/` contains a directory `foo/` and a file `file4`.

Paths

A *path* is used to give the location of a file or directory one is interested in. There are two types of path:

- An *absolute path* gives the path to a file or directory starting from the root path, indicated by `/`. Hence a path such as `/home/njn2/foo/file3` is an absolute path. An absolute path is an unambiguous way to indicate where a file or directory can be found.
- A *relative path* gives the path to a file or directory relative to some directory. For instance, relative to the directory `/home/njn2`, the path to the file `file3` in directory `foo/` is given by `foo/file3`.

Relative paths are always relative to a directory called the *current working directory*, or simply the *current directory*. When you login, your current directory is your home directory, `/home/`*`username`*`/` on `turing`.

The following sequences of characters have special meaning in a path.

- The sequence `.` indicates the directory the path points to. For example, the path `/home/./njn2/` is equivalent to the path `/home/njn2/`, which is equivalent to the path `/home/njn2/`, The relative path `.` is simply the current working directory. Thus, if the current directory is `/home/`, `.` is equivalent to just `/home/`.
- The sequence `..` goes up one directory. Thus if the current directory is `/home/njn2/`, the path `../cs114/` indicates the directory `/home/cs114/`. Similarly, the path `/home/njn2/../cs114/` also indicates the directory `/home/cs114/`.
- A shortcut way to refer to anyone's home directory is to use the special prefix `~`*`username`*. This is an absolute path, equivalent on `turing` to `/home/`*`username`*. For example, the home directory of user `njn2` can be referred to as `~njn2/`. To refer to your own home directory, you can just use `~`. Thus, `~/foo/bar` refers to the file `bar` in directory `foo` in your home directory.
- To refer to your home directory, you can also use the *environment variable* `$HOME`. We'll discuss environment variables later in the course.

Commands

cd

You can change your current working directory by using the command `cd` at the prompt:

```
% cd path
```

changes the current working directory to *`path`*. (I will often indicate commands to be typed at the prompt by prefixing the line with a `%`; don't type the `%`.) The *path* argument can be either an absolute or relative path. Thus,

```
% cd /
```

changes to the root directory;

```
% cd /home/njn2/
```

changes to the home directory of `njn2`;

```
% cd ..
```

changes to the *parent directory* of the current working directory.

If you just type

```
% cd
```

by itself, you will change back to your home directory.

pwd

You can print your current directory using the command `pwd`, which stands for *print working directory*. For example:

```
% pwd
/home/njn2
```

indicates that the current directory is `/home/njn2/`.

ls

The command `ls` will display the contents of the current directory. Thus, if the current directory is `/home/njn2/`:

```
% ls
file3
foo
```

If you give a directory argument to `ls` it will list the contents of that directory:

```
% ls ~cs114
file1 file2
```

```
% ls ~cs114 ~njn2
/home/cs114:
file1 file2

/home/njn2:
file3 foo
```

Filesystem manipulation

So far we can navigate the filesystem and see what's there. Now, how do we manipulate the filesystem. The following commands are also useful for dealing with files and directories. Recall than whenever we talk about a file or a directory, it actually stands for a path to a file or directory.

- `mkdir` *`directory`* – creates a *`directory`*
- `rmdir` *`directory`* – removes a *`directory`* (only if it's empty and you're not inside it)
- `touch` *`file`* – creates an empty *`file`*
- `cat` *`file`* – dumps the contents of *`file`* to the console
- `rm` *`file`* – removes *`file`*
- `mv` *`oldname newname`* – renames (moves) a file
- `mv` *`file1 file2 ... directory`* – moves files into a directory
- `cp` *`oldname newname`* – copies a file

- `cp file1 file2 ... directory` – copies files into a directory

Some commands have *options*, which affect the way they behave. Options are typically a letter preceded by a `-`. For example, if you give the option `-i` to `rm`, that is, if you write `rm -i file`, the system will prompt you for confirmation before deleting the file.

```
% rm -i file
rm: remove file (yes/no)?
```

Some commands understand many options. The command `ls` for instance, recognizes, among others, the option `-F`, which makes `ls` give you some indication of the type of each file.

```
% ls -F file3 dir/
```

It appends a `/` at the end of every directory. `ls` also takes the option `-a`, which makes `ls` display *everything* in the directory, including so-called *hidden* files. A file or directory is hidden if its name starts with a dot (`.`), for example `.hidden`. Hidden files aren't really special in any way, except that `ls` and other tools won't list them by default. Typically, programs needing configuration files will make those files hidden; otherwise, these files will clutter up your home directory.

man

How do you learn and remember the various options that each command understand? You can look at the online Unix documentation, available through the command `man`. If you type:

```
% man command
```

it will search for and display documentation (the **man**ual) on *command*. This information is called the *man page* of the command. Among other things, you will get a description of the arguments the command expects and the options it understands. The command `man` itself understands different options; do:

```
% man man
```

to find out what those options are.