```
#!/bin/bash
echo "what is your name?"
read name
echo "Hello $name!. Welcome to ICT Skills Scripting Tutorial"
```

Some Utilities

The Arrow Keys

Suppose you want to run a command that you ran a short while ago? Within the terminal window, this is accomplished using the `up-arrow` key to retrieve previous commands.

Once you have viewed some earlier commands, note that the`down-arrow` also works, which can be helpful if you go too far.

Type `pwd` to see print your current working directory
Type `ls` –l to get a list of the files here
Type `clear`

The `history` Command

But what if the command you want was several commands ago, say 10? Who wants to type that many arrows?

Type `history`
In this listing, each command is numbered for future reference.

For example, with this listing, if you want to re-issue a previous `pwd` command, you could do so by typing`"!20"` (or whatever number it is!) to repeat command number 20. Better yet (since you don't need to know the command number)

Type `!p` #This will re-issue the most recent command that begins with the prefix "p".
Type `!pw` **Note** the prefix can be any length; it is not limited to one character i.e. `!pw`

`cat`

It is often convenient to look at the contents of a text file without having to open it in an editor. The simplest command for this purpose is `cat`. For example,

Type `cat .bashrc`
would display the contents of the .bashrc file in your terminal window. Although `cat` works well for short files that can be viewed all on one screen, we will see shortly that other commands may work better for longer files.

`.bashrc` #runs whenever it is started interactively. rc was inspired by the run com facility from early Unix releases
`cat` #stands for "concatenate" (which means "to combine" or "to join together")

Type `cat` #without any command, you're only moved to the next line, where the system is waiting for some standard input

<Ctrl> + c is the universal signal for Cancel in Linux.

Wildcards

Wildcards are a set of building blocks that allow you to create a pattern defining a set of files or directories. As you would remember, whenever we refer to a file or directory on the command line we are actually referring to a path. Whenever we refer to a path we may also use wildcards in that path to turn it into a set of files or directories.
Here is the basic set of wildcards:

- ***** - represents zero or more characters
- **?** - represents a single character
- **[]** - represents a range of characters (these are the square brackets)

Further Reading, Study and Practice on http://ryanstutorials.net/bash-scripting-tutorial/

E.g. Using wildcards, what command would

1. remove all Java files?
2. remove all java files with three letters?
3. remove all files with three letters,irrespective of their extension, if there is any?
4. remove all files whose second letters is i?
5. list all files beginning with a b?
6. List every file whose name either begins with a c or m?
7. List every file whose name includes a digit?

Directory and File Commands

The Linux Directory/File Hierarchy

Linux maintains directories and files in a hierarchical structure, called a *tree structure*. This section explores this organization.

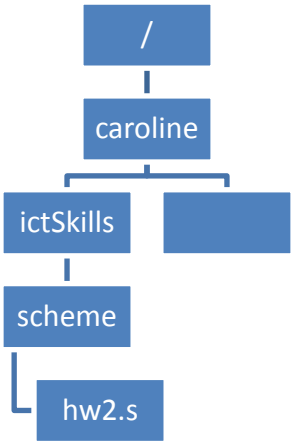
Pathnames

When you first open a terminal window, the shell expects that you are in your home directory. At the start, this is called your current "working directory" (i.e., your position within the directory tree).

A *relative pathname* for a file is a name that is given "relative" to your current working directory.

For example, if your current working directory is `ictSkills`, then `scheme/hw2.s` could be a relative pathname for a file named `hw2.s` located in a directory named `scheme` that was itself inside `ictSkills`.

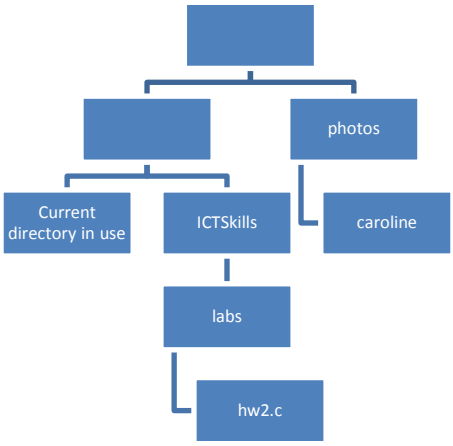
An *absolute pathname*, such as `/home/username/ictSkills/scheme/hw2.s`, includes the file's complete path starting with the system's "root" directory, which is always named "`/`" on a Linux system. Just like it sounds, the root directory is the topmost directory in the file system tree.



Each directory in a Linux system contains two special files "`.`" and "`..`" that can be useful when constructing relative pathnames.

"." and ".."

The file named "`.`" means "the current directory," and the file named "`..`" means "the parent directory".

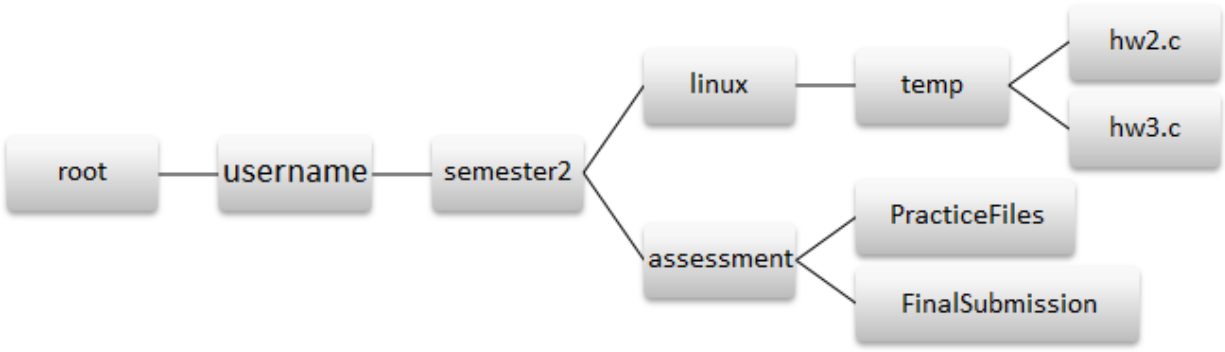


`../ICTSkills` could be a directory that is a sibling of your current working directory (i.e., `ICTSkills` could be a directory that has the same parent directory your current working directory does).
`../ ICTSkills /labs/hw2.c` could refer to a file "hw2.c" that resides farther down that branch of the tree.

`../../photos/caroline` could be a directory that is a cousin of your current directory in the file system tree.

EXERCISE: Recreate the following structure, using `m`_____ command to create a directory and **`t`**_____ to create a blank file (without opening it)

mkdir -p semester2/linux/temp/hw2.c semester2/linux/temp/hw3.c
mkdir -pv semester2/assessment/PracticeFiles
What is the –p option for? _____
What is the –pv option for? _____

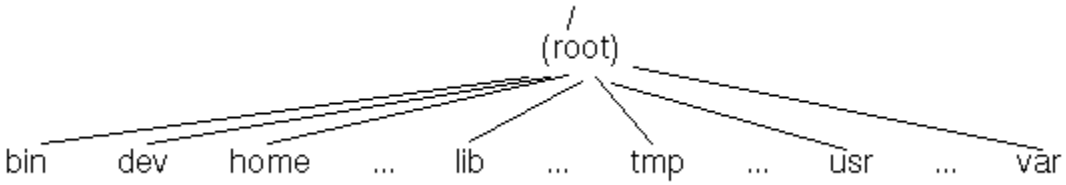


~ character

The tilde character is also useful for specifying pathnames, but it works a little differently. Used alone, it specifies your home directory, so `~/semester2/linux` is a short name for `/home/username/ semester2/linux`.
Finally, `~username` refers to the home directory belonging to *username*. Thus, you can print a listing of the files in my `public_html` directory with
Go to the temp directory
Type ls #this is the letter "L"
Type ls -l ~username/semester2/assessment #replace username with your own username
Type cd ~
Type pwd

Root Directory and its Subdirectories

While we are poking around the Linux file system, take a look at the files in the root directory `/`. You should see directories with names like `/bin`, `/home`, `/lib`, and`/usr`.



Again, list the files in each of these directories. They contain many, many files, organized as follows.

- `/bin`: These are the executable programs that comprise the GNU/Linux utilities. For example, there is an executable file here named `ls` that is run when you issue the command `ls`.
- `/home`: You won't be surprised to hear that user accounts are stored in this directory.
- `/lib`: This directory is the home of many libraries that can be used by programmers. For example, you should be able to find a file named `libc-2.3.6.so` here, that contains the "standard c library functions".
- `/usr`: Generally contains application programs, libraries, and other files that are not part of the GNU/Linux system (i.e., optional resources intended for and requested by users). For example, the acrobat reader is located here under `/usr/bin/acroread`.

File Utilities

Some common file management commands are listed in the table below. You should try each of these to determine just how they work.

Utility	Description
`ls`	"list" files and directories
`pwd`	"print working directory"
`cd`	"change (your working) directory"
`mkdir`	"make directory"
`rmdir`	"remove directory"
`cp`	"copy" a file or directory
`mv`	"move" a file or directory (i.e., rename it)
`rm`	"remove" a file (i.e., delete it)

The following variants can be particularly handy. (Although some details may not be obvious now, we will see shortly how to find out more information about such commands.)

```
cd ..          cp            ls -l         mkdir -p
```

EXERCISES:

Copy hw2.c to the same folder to be named hw4.c

Move FinalSubmission into the linux folder

Copy the temp directory and files to a new directory called CA1

> In it's default behaviour cp will only copy a file. Using the **-r** option, which stands for recursive, we may copy directories.

Delete the temp directory

The Linux command line does not have an* undo *feature.
Perform destructive actions carefully.

pushd and popd

Two more commands that can be quite useful for moving around the file system are `pushd` and `popd`. They let you jump back and forth between distant directories quickly and easily. For example, suppose your current working directory is `semester2/assessment/PracticeFiles` and you want to jump to a directory in another branch of your file system tree, say `semester2/linux`, and afterward you want to return to your original directory.

The following command will push the name of your current directory onto a stack of directory names that Linux maintains behind the scenes, and then change your current directory to the one named in the command:

```
pushd ~semester2/linux
```

When you are ready to return to your previous directory, you simply type `popd`. This pops the most recent directory name off the stack and then makes it your current working directory.
If you like, you can use `pushd` several times in a row (pushing multiple directory names onto the stack), and then backtrack through the sequence in reverse order.
As one application, you might use `pushd` and `popd` when jumping back and forth between labs or homework assignments.

Displaying Text Files

It is often convenient to look at the contents of a text file without having to open it in an editor. Previously in this lab, we saw that `cat` can be used for this purpose, but it is most useful for short files that can be viewed all on one screen.
GNU/Linux provides several other utilities that are useful for "paging" through text files (i.e., for viewing files one page at a time). Several of these commands are outlined in the following table.

Command	Description
`more`	move through a file screen by screen (hit space for the next page, return for one more line)
`less`	a new and improved version that allows backward paging as well with the `up-arrow`, `down-arrow`, `Page Up`, and `Page Dn`.
`head`	show the first few lines of a file
`tail`	show the last few lines of a file

The Manual man pages

Linux includes an on-line help system, called the *manual*. There is a "man page" describing each Linux command

Type **man cat** to read about `cat` command.

> You should see the command name and a quick synopsis of how to use the command. For example, the synopsis"`cat [OPTION] [FILE] ...`" tells you that `cat` (optionally) takes a file as it's input, and it also can take (optional) options as parameters. A list of the options follows the synopsis.

Another handy use for the man pages is finding commands when you don't remember, or never knew, their names.

> Suppose you wanted to find a function for computing the square root: you could guess that such a function might exist, but you might not know its name. To print a list of man pages that include the word "square" in the name or description fields, you could use "`man -k square`".

Directory and File Permissions

From a user's perspective within Unix or Linux, the world of files and directories is divided into three categories:

- the user
- the user's group (e.g., ComputerScience faculty or student)
- everyone else

For each of these categories, users in these categories can have three types of capabilities:

- an individual might have permission to **read** a file
- an individual might have permission to **write** or modify a file
- an individual might have permission to **execute** the file (e.g., run a program)

To clarify how these permissions work, we consider a long listing of files in a subdirectory for Caroline Cahilll:

```
ls -l ~cahill/public_html
total 40
drwxr-xr-x 4 cahillcmathfac 4096 2017-04-25 16:32 csc105
drwxr-xr-x 6 cahillcmathfac 4096 2017-01-19 16:04 csc152
-rw-r--r-- 1 cahillcmathfac 5808 2017-09-01 20:38 index.html
drwxr-xr-x 2 cahillcmathfac 4096 2017-01-02 15:55 mmc_files
```

Setting Permissions

You can set the permissions of the files you own using the `chmod` command. The simplest approach is to assign numbers to each capability (4 for read, 2 for write, 1 for execute) and then to use addition when combining numbers. Thus, 6 = 4+2 (read plus write permission), and 7 = 4+2+1 (all three permissions added together).

Within this framework, you set permissions for a file by specifying the desired capabilities for the user, group, and world (in that order). Thus, when she set up her directory for `assessment` above, might have issued the command

`chmod 755 assessment`

> Here, the user (C Cahill) has full permissions (7=read+write+execute); while the others can read and execute, but not write (5).