LINUX Tutorial 3

Basic Directory (Folder) Manipulation

You can create directories and subdirectories to organize your files, just as you can create folders in a graphical user interface (GUI). It is recommended that you create a directory for each class and store all your files for that class in that directory to keep your files organized.

When you first login, you are automatically put into your "home" directory, which is your user-specific file space on Odin. Odin does not remember what directory you were in when you logged off, so if you want to store all of your class files in a specific directory, you have to remember to go to that directory every time you login.

The following commands are used to create and manipulate directories, and also to move data between directories. A demonstration of these commands will be given on the projector:

pwd	Prints the name of the directory you are currently in.
mkdir <dir_name>	Create a directory with the given name.
rmdir <dir_name>	Remove the directory with the given name. The directory must be empty first.
ls [-la] <dir_name>	List the files contained in the given directory.
ls	List the files in your current directory.
cd <dir_name>	Change your current directory to the given name.
cd	Change your current directory to your home folder.
cp <filename> <dir_name>	Copy the given filename into the given directory.
mv <dir_name> <new_name>	Rename a directory
mv <filename> <dir_name>	Move (not copy) the given file into the given directory.

Understanding File Permissions

Since Unix is a multiuser operating system, all files and directories have permission flags to control access.
A class demonstration will be given demonstrating what these permissions are, and how to edit them.

Managing Processes

A process is an executing program under Unix/Linux. Each process has an owner (generally the owner of the executable file) and a unique process ID known as the PID. As the owner of a process, you can tell Odin to stop the process, which is known as "killing" the process. You can only kill processes that you own, so don't be concerned that a mistake might crash the server (see man kill). As you begin writing programs, you often need to use the kill command to kill a program stuck in an infinite loop or a frozen shell process.

The first way one can control a process is to send a control character to the currently running process (the foreground process). For example, if your program is stuck in an infinite loop, you can try to send the control character for kill. Control characters are initiated by holding the CTRL key and hitting another character.

The common control characters are:

CTRL + C	Sends a kill signal for the current foreground process
CTRL + Z	Suspends the current foreground process
CTRL + D	Sends an EOF signal to the foreground process

Some processes can ignore the control signals, so the control characters will not always work. If the control character does not work, you will need to use the kill [-9] <PID> command. In order to use this command, you first need to discover the PID for your out of control process. To do this, you will use the ps command.

The ps command has many options that affect which processes it shows you. By default, it will just show you the processes for your current login session. But if one login is stuck in an infinite loop, you will need to start a second login session to kill the stuck process.

To view all of your processes, use one of the following commands:

ps x	Shows all processes owned by you
ps ux	Shows detailed information for your processes
ps -ef | grep <username>	Filters all processes, looking for a username
ps ux > processes.txt	Writes details information for your processes to the file processes.txt

Once you have the list of processes, you need to look for the line that contains your runaway process. The number towards the start of the line (under the PID column) is the PID for the process. We can then give that to the kill command.

For example, if the PID is 12345, the kill command would be one of the following:

kill 12345	Politely request that the process exit. Can be ignored.
kill -9 12345	Demand that the process exit immediately. Not ignored.

One final note on processes. Foreground processes can be sent to the background when you want to keep using the shell prompt for other tasks. This is useful when the program does not need user interaction.

To start a program in the background, append an ampersand (&) to the command:

cat &	Start cat in the background

Make sure that the program can complete in the background.
Otherwise, it will sit there and do nothing as it waits for the user to type something.

To see all your current background processes, type the command

```
jobs
```

To bring a background process back to the foreground, use the command

```
fg <job_number>
```