File Management Commands

- **mkdir *directory name*** – create a new directory
- **touch *file name*** – create a new file with no content or touch an exisiting file
- **cd *directory name*** – change directory to the one mentioned

  The directory name can be relative to where you are or absolute. Suppose you are in the directory /home/nolan/stat133, and it contains a directory notes. The next three commands all change the directory to /home/nolan/stat133/notes.

  – **cd notes** – The directory name is relative to the current directory
  – **cd /home/nolan/stat133/notes** – Here we have supplied the full pathname to find the directory
  – **cd ~/stat133/notes** – We use the symbol ~ to refer to the user's root directory, which in this example is /home/nolan.
  – **cd ../** – This relative change is to the directory above the current directory, i.e. to /home/nolan/stat133
  – **cd ../stat205** – This relative change is to the directory at the same level of the tree as the current directory i.e. /home/nolan/stat205

File Management Commands - Ctd

- **rmdir *directory name*** – remove directory (it must be empty to be removed
- **pwd** – gives you the name of the current (working) directory
- **cp *file1 file2*** – copy the first file to the second file
- **mv *file1 file2*** – rename (i.e. move) the first file to the second file name
- **rm *filename*** – remove the file
- **ls** – list the files and directories in the current directory
- **more *filename*** – shows the contents of a file
- **tail *filename*** – shows the last lines of a file

Example

Let's create a system of files and directories that matches the handout from Friday.

```
> mkdir Test
> cd Test
Test> touch x
Test> touch file1.tex
Test> mkdir A
Test> mkdir A/B
Test> cd A
Test/A> touch file2.tex
Test/A> touch file3.csv
Test/A> mkdir C
Test/A> touch C/x
Test/A> touch B/file4.doc
Test/A> touch B/file5.txt
Test/A> mkdir B/D
Test/A> touch B/D/x
Test/A> touch B/D/x2
```

ls – list directory contents

From within the Test directory that we set up we see the directory A and the two files x and file1.tex.

```
> ls
A  myfile.tex  x
```

The **-R** option say recurse your way through the tree of subdirectories excuting the **ls** command as you go.

```
> ls -R
.:
A  file1.tex  x
./A:
B  C  file2.tex  file3.csv
./A/B:
D  file4.doc  file5.txt
./A/B/D:
x  x2
./A/C:
x
```

Help for commands

man ls – the online manual pages for the **ls** command

ls - -help – abbreviated help on the various options to the **ls** command.

```
Usage: ls [OPTION]... [FILE]...
List information about the FILEs (the current directory by default).
Sort entries alphabetically if none of -cftuSUX nor --sort.

Mandatory arguments to long options are mandatory for short options too.
-a, --all                  do not hide entries starting with .
-A, --almost-all           do not list implied . and ..
    --author               print the author of each file
-b, --escape               print octal escapes for nongraphic characters
    --block-size=SIZE      use SIZE-byte blocks
-B, --ignore-backups       do not list implied entries ending with ~
-c                         with -lt: sort by, and show, ctime (time of last
-C                         list entries by columns
    --color[=WHEN]         control whether color is used to distinguish file
-d, --directory            list directory entries instead of contents,
-D, --dired                generate output designed for Emacs' dired mode
-f                         do not sort, enable -aU, disable -lst
-F, --classify             append indicator (one of */=@|) to entries
...
```

* – the wild card

The `*` symbol matches any number of characters (except the /). It can be quite handy when looking for files that have particular type.

Below we list only those files with the filetype extension of `tex` in the subdirectory A:

```
> ls A/*.tex

A/report.tex

> ls -R *.tex

ls: No match.
```

Can you explain why the second command does not find any files with the filetype `tex`?

Piping |

We can construct more complex shell commands by piping or sending the output from one comand to the input of another. Below we pipe the output from the **ls** command to the **wc** comand.

```
> ls -R | wc

  33      29     140
```

wc – short for word count, returns the number of newlines, words, and bytes in the input file. Does this mean that there are 33 files in the Example directory?

find – search for files in a directory hierarchy

The **find** command may be better suited to the previous task.

```
> find -name '*.tex'

./A/report.tex
```

We can also do fancier finds, such as all tex files in my home area that have been modified within the past 21 days.

```
> find /home/nolan/ -mtime +21 -name '*.tex' | wc -l

2421
```

Or, we can find all those files that do not end with .tex in the working directory.

```
> find . -type f -not -name '*.tex'
```

grep – print lines matching a pattern

With **grep** we can search for patterns within a file. The syntax is
grep [options] PATTERN [FILE...].

Suppose we want to find all files that have the words ”data frame” in them.

```
> grep -lR 'data frame' *

DataTypes/DataTypes.tex
DataTypes/Assignment.tex
DataTypes/Lists.tex
Introduction/introduction.tex
ShellCmds/.ShellCmds.tex.swp
Traffic/traffic.tex
schedule
```

grep continued

To invert the match we can use the **-v** option to the **grep** command.

```
> grep -lRv 'data frame' * | more
DataTypes/DataTypes.tex
DataTypes/htmlPDF.pdf
DataTypes/DataTypes.log
DataTypes/DataTypes.aux
DataTypes/DataTypes.pdf
DataTypes/pdfFiles.tex
...
DataTypes/ExampleLatex2.log
DataTypes/ExampleLatex2.aux
DataTypes/ExampleLatex2.pdf
--More--
```

Here the **more** command formats the output being sent to the console so that you see one page at a time, rather than have the whole output go whizzing past.
To proceed to the next page of output hit the space bar.
To stop displaying the output hit **q**

Other useful aspects of the Shell commands

- **$>$ and $>>$** – redirection
  At times we want to save the output from a command to a file. We can do this by redirecting the output to a file. We use the single $>$ to direct the output to a new file (it will also overwrite an existing file), and we use the double $>>$ if we want to add the output to an existing file.
- **chmod** – change permissions on files and directories.
  In Unix we can define groups of users, such as all student accounts for Stat133, or all faculty accounts, or all graduate studnet accounts. The **chmod** command allows us to set specific permissions for the owner of the file, the group that the owner belongs to, and for all other accounts. The **-l** option on the **ls** command provides the permissions.

```
Test> ls -l
total 4
drwxr-xr-x    4 nolan    nolan        4096 Sep 19 11:54 A
-rw-r--r--    1 nolan    nolan           0 Sep 19 11:53 file1.tex
-rw-r--r--    1 nolan    nolan           0 Sep 19 11:53 x

Test> chmod g+w *
Test> ls -l
```

```
total 4
drwxrwxr-x    4 nolan    nolan        4096 Sep 19 11:54 A
-rw-rw-r--    1 nolan    nolan           0 Sep 19 11:53 file1.tex
-rw-rw-r--    1 nolan    nolan           0 Sep 19 11:53 x
```

- **sort** – sort lines of text files
- **uniq** – remove duplicate lines from a sorted file
- **cat** – concatenate files and print on the standard output
- **tail** - output the last part of files

Process controls

- **ps** - report a snapshot of the current processes.
- **kill** - send a signal to a process

More advanced shell techniques

- **sed** –a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline).
- It is possible to write shell scripts, i.e. programs with loops and variables, and excute these at the command line