A Crash Course on UNIX

- UNIX is an "operating system".
  - Interface between user and data stored on computer.
- A Windows-style interface is not required.
- Many flavors of UNIX (and windows interfaces).
  - Solaris, Mandrake, RedHat (fvwm, Gnome, KDE), ...
- Most UNIX users use "shells" (or "xterms").
  - UNIX windows systems do provide some Microsoft Windows functionality.
The Shell

• A shell is a command-line interface to UNIX.
  – Also many flavors, e.g. sh, bash, cs, csh, tcs.

• The shell provides commands and functionality beyond the basic UNIX tools.
  – E.g., wildcards, shell variables, loop control, etc.

• For this tutorial, examples use tcs in RedHat Linux running Gnome.
  – Differences are minor for the most part...
Basic Commands

• You need these to survive: `ls`, `cd`, `cp`, `mkdir`, `mv`.
  - Typically these are UNIX (not shell) commands.
  - They are actually programs that someone has written.
  - Most commands such as these accept (or require) "arguments".
    • E.g. `ls -a` [show all files, incl. "dot files"]
    `mkdir ASTR688` [create a directory]
    `cp myfile backup` [copy a file]

• See the handout for a list of more commands.
A Word About Directories

• Use `cd` to change directories.
• By default you start in your home directory.
  – E.g. `/home/dcr`
• Handy abbreviations:
  – Home directory: `~`
  – Someone else's home directory: `~user`
  – Current directory: `.`
  – Parent directory: `..`
Shortcuts

- To return to your home directory: `cd`
- To return to the previous directory: `cd -`
- In `tcsh`, with filename completion (on by default):
  - Press `TAB` to complete filenames as you type.
  - Press `Ctrl-D` to print a list of filenames matching what you have typed so far.
  - Completion works with commands and variables too!
- Use `↑, ↓, Ctrl-A, & Ctrl-E` to edit previous lines.
Man Pages

• To see all possible options to a command, use the man command, e.g. man mv.

• WARNING: the man pages are very terse...
  – Not for the novice; get a book instead, or go surfing.

• You can search the man pages by keyword with the -k option.
  – E.g. man -k rename

• Sometimes a command provides its own help.
Wildcards

- Wildcards provide handy filename substitution.
  - E.g. `ls *.c` [list all files with extension ".c"]
- In tcsh, square brackets substitute for a range.
  - E.g. `cp obs0[0-9].fits tmp` [copy first 10 FITS files]
- Curly brackets can be used to repeat patterns.
  - E.g. `a{b,c,d}e` is shorthand for `abe ace ade`
- Use `\` or single quotes (') to disable substitution.
  - E.g. `cd Data\[Oct01\]` or `cd 'Data[Oct01]'`
Stream Redirection

- Normally commands expect to receive input from the keyboard and/or send output to the screen.
- Special redirection symbols can override this.
  - E.g.  `ls > files.txt`  [send listing to file]
    `mail dcr < hwk`  [mail file to user dcr]
    `ls -l | more`  [pause listing by screenfuls]
- There are many other examples: see handout.
- **WARNING**: the syntax is very shell dependent!
Shell Variables & Aliases

- You can store information in a shell variable.
  - E.g. `set work = /home/dcr/Work`

- To access the info, prepend a dollar sign ($).
  - E.g. `cd $work`

- Shell variables are local to the shell; environment variables are inherited by new shells and can even be accessed internally by programs.
  - E.g. `setenv WORK /home/dcr/Work`
Shell Variables & Aliases, Cont'd

• There are certain special variables.
  – E.g. PATH contains a list of directories to search for commands

• Aliases allow you to define new commands.
  – E.g. alias rm rm -i  [make rm ask for confirmation]

• Variables and aliases that you use all the time can be defined in your "startup" file.
  – E.g. in tcs h, ~/.tcs hrc is your startup script
Command Substitution

- In tcsh, you can use the result of a command as part of a command.
  - E.g. setenv OS `uname`
- Anything inside backward single quotes is first evaluated in its own shell, and the result is returned as a string of one or more words.
- This is very handy in scripts and in conjunction with tools like `sed` and `awk`. 
A Quick Word on Editors

• There are many text editors to choose from.
  – E.g. *vi*, *emacs*, *pico*, etc.

• To create scripts or programs, you will need to learn how to use an editor!
  – Also essential if you want to use formatting tools such as *LaTeX*, etc.

• Windows systems often have good GUI editors.

• Note you can use *cat* or *more* to show file data.
The "stream editor" (sed) is a useful tool for changing the contents of a file (or stream).

- E.g. sed s/apples/oranges/ myfile.txt will change the first occurrence of "apples" on each line of myfile.txt into "oranges". To change every occurrence, do the following: sed s/apples/oranges/g myfile.txt.

sed is great in scripts, but it can also be used from the command line. E.g., in conjunction with the foreach command, it's a handy way to rename lots of files, like all *.JPEG files to *.jpg (EFTS).
• **awk** is a powerful "pattern scanning and processing" language.

• Use it to print a column of a file:
  – E.g. `awk '{print $2}' myfile.txt` [print 2\textsuperscript{nd} column]

• Use it to do math:
  – E.g. `awk '{print $1+$2}' myfile.txt` [add columns]

• Use it as a calculator:
  – E.g. `echo '' | awk '{print sqrt(2)}'`
You can write entire programs in awk:

- E.g. awk '/error/ {print $0; n += 1} END {print n}'
  myfile.txt  [counts and displays lines containing "error" in file]

Like tcsh itself, awk syntax is reminiscent of the programing language C.

awk, sed, wildcards, shell variables, stream redirection, and command substitution enable the creation of very sophisticated tcsh scripts...
A script is a sequence of shell commands, usually stored in a file and either sourced or executed like a program. Here's a simple example:

```
foreach file (*)
    if (-d $file) then
        echo $file is a directory
    endif
end
```

To aid with scripting, `tcsh` has a number of built-in commands, such as `foreach`, `if`, `while`, etc.
A special variable called `argv` is defined inside a script (shell). It contains any arguments passed to the script (shell).

- E.g. `echo $argv` [show all arguments]
  `echo $argv[2]` [show the 2\textsuperscript{nd} argument]
  `echo $#argv` [show the number of args]

You can do integer math within a script using `@`.

- E.g. `set x = 0; @ x = $x + 1; echo $x` [good for loops!]
What We Didn't Cover

- File permissions (chmod)
- Managing jobs (ps, nice, kill)
- Printing (lpr)
- Remote connections (ssh, scp)
- System administration (not for the faint of heart)
- And lots of other stuff!
  - See the handout for web tutorials, etc.