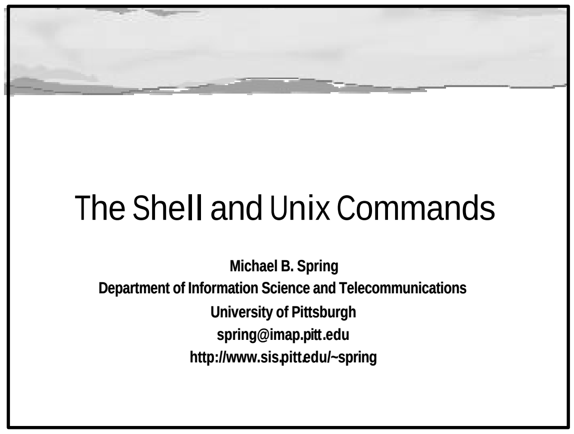
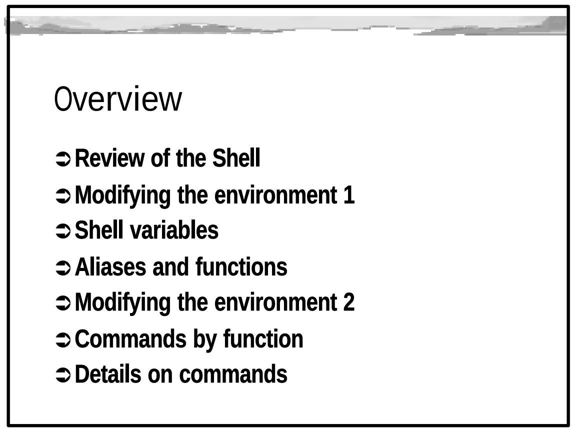


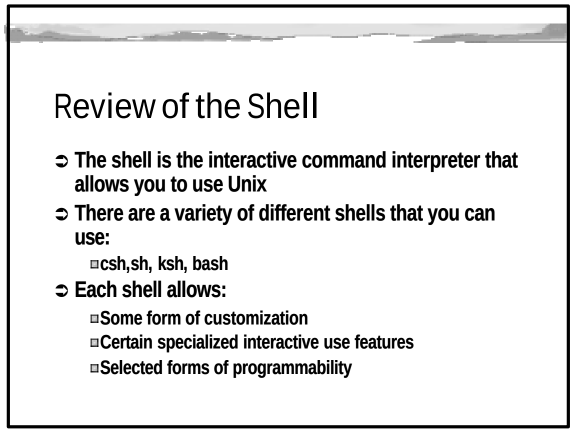
The Shell and Unix Commands

Michael B. Spring
Department of Information Science and Telecommunications
University of Pittsburgh
spring@imap.pitt.edu
http://www.sis.pitt.edu/~spring



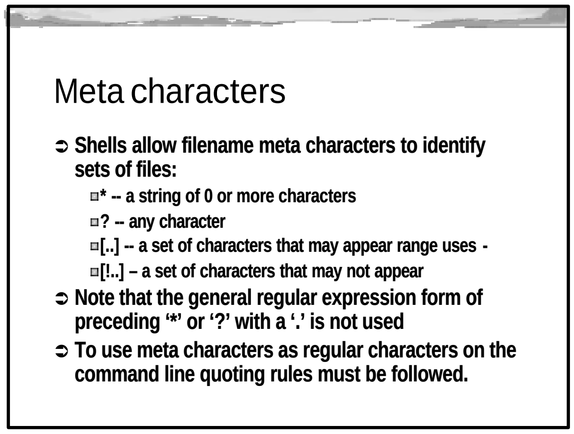
Overview

- ➲ Review of the Shell
- ➲ Modifying the environment 1
- ➲ Shell variables
- ➲ Aliases and functions
- ➲ Modifying the environment 2
- ➲ Commands by function
- ➲ Details on commands



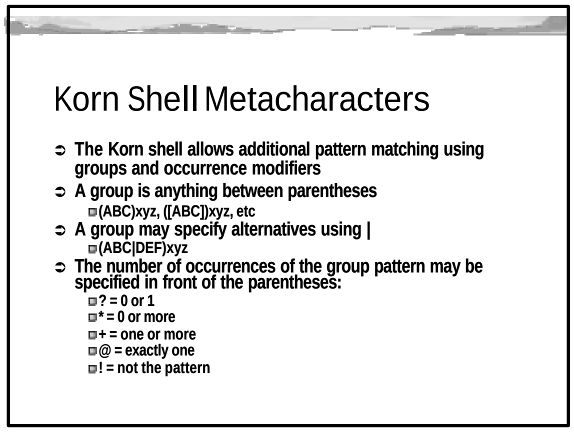
Review of the Shell

- ➲ The shell is the interactive command interpreter that allows you to use Unix
- ➲ There are a variety of different shells that you can use:
  - ◻csh,sh, ksh, bash
- ➲ Each shell allows:
  - ◻Some form of customization
  - ◻Certain specialized interactive use features
  - ◻Selected forms of programmability



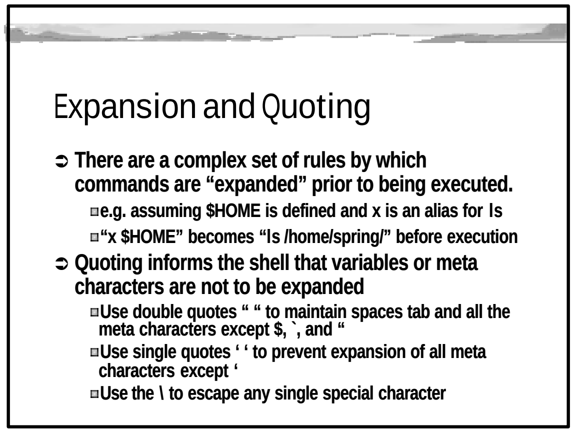
Meta characters

- Shells allow filename meta characters to identify sets of files:
  - * -- a string of 0 or more characters
  - ? -- any character
  - [..] -- a set of characters that may appear range uses -
  - [!..] – a set of characters that may not appear
- Note that the general regular expression form of preceding '*' or '?' with a '.' is not used
- To use meta characters as regular characters on the command line quoting rules must be followed.



Korn Shell Metacharacters

- The Korn shell allows additional pattern matching using groups and occurrence modifiers
- A group is anything between parentheses
  - (ABC)xyz, ([ABC])xyz, etc
- A group may specify alternatives using |
  - (ABC|DEF)xyz
- The number of occurrences of the group pattern may be specified in front of the parentheses:
  - ? = 0 or 1
  - * = 0 or more
  - + = one or more
  - @ = exactly one
  - ! = not the pattern



Expansion and Quoting

- There are a complex set of rules by which commands are "expanded" prior to being executed.
  - e.g. assuming $HOME is defined and x is an alias for ls
  - "x $HOME" becomes "ls /home/spring/" before execution
- Quoting informs the shell that variables or meta characters are not to be expanded
  - Use double quotes " " to maintain spaces tab and all the meta characters except $, `, and "
  - Use single quotes ' ' to prevent expansion of all meta characters except '
  - Use the \ to escape any single special character

Back quotes – lower case ~

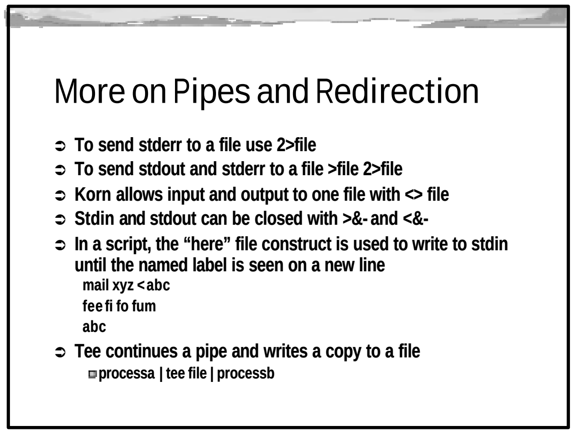
- Back quotes -- ` ` are used to substitute the results of a command in line
  - xx==`ls`, would set xx equal to the listing of files in the current working directory
  - A backquoted string will be used frequently in scripts to build a set of files (using ls in a for set)
  - Imagine running a program x that required fully qualified pathnames for the input and output files
    - x –i `pwd`/infile –o `pwd`/outfile

Process control

- When Unix executes an external command, the shell waits until the process completes before providing an additional prompt.
- A process can be run in the background by following the command with an &
  - Alternatively, a running process can be suspended (^Z) and then placed in the background with the command bg. (^D will kill a running process.)
- Multiple processes can be run sequentially through one input line by separating them with a ;
  - Commands on a single line can also be grouped inside ()

I/O, Pipes and redirection

- Each process in Unix has access to file handles that allow input and output.
- Each process starts with the handles 0, 1, and 2 assigned for stdin, stdout, and stderr
- Processes written to read and write stdin and stdout may be "piped" on the command line with |
- The input to a process may be redirected from a file (<). Output may be redirected with a >. (e.g. >file)
- Output may be appended to a file with >>

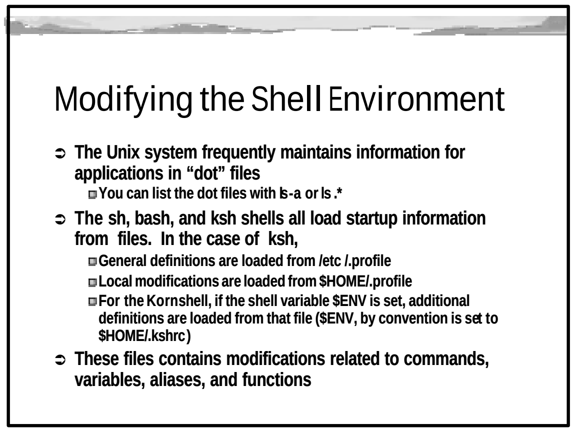


More on Pipes and Redirection

- To send stderr to a file use 2>file
- To send stdout and stderr to a file >file 2>file
- Korn allows input and output to one file with <> file
- Stdin and stdout can be closed with >&- and <&-
- In a script, the “here” file construct is used to write to stdin until the named label is seen on a new line

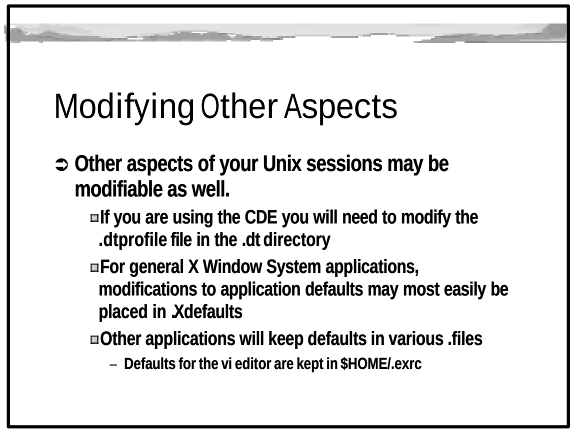
```
mail xyz <abc
fee fi fo fum
abc
```

- Tee continues a pipe and writes a copy to a file
  - processa | tee file | processb



Modifying the Shell Environment

- The Unix system frequently maintains information for applications in “dot” files
  - You can list the dot files with ls -a or ls .*
- The sh, bash, and ksh shells all load startup information from files. In the case of ksh,
  - General definitions are loaded from /etc /.profile
  - Local modifications are loaded from $HOME/.profile
  - For the Kornshell, if the shell variable $ENV is set, additional definitions are loaded from that file ($ENV, by convention is set to $HOME/.kshrc)
- These files contains modifications related to commands, variables, aliases, and functions



Modifying Other Aspects

- Other aspects of your Unix sessions may be modifiable as well.
  - If you are using the CDE you will need to modify the .dtprofile file in the .dt directory
  - For general X Window System applications, modifications to application defaults may most easily be placed in .Xdefaults
  - Other applications will keep defaults in various .files
    - Defaults for the vi editor are kept in $HOME/.exrc

Shell Variables

- The shell allows the user to introduce variables that have values. Keep in mind that the value is always a string. (Actually Korn allows integer variables.)
- It is easy to set a variable
  - varname=value
- A variable available to spawned processes is an environment variable. To create one, export it:
  - Export varname
- There are many variables important to the shell:
  - Standard Variables
  - Built-in variables

Variable Basics

- No spaces in the set
  - MBS=Michael
- Use quotes to allow spaces
  - MBS=" Michael B. Spring"
- Refer to a variable using $, or more formally ${varname}
  - echo $MBS
- The Korn shell allows arrays
  - set –A MBS 23 45 67 93 42
  - Formal syntax required -- ${MBS[0]} = 23, ${MBS[3]} = 93
- Use set, unset, and typeset to control variables – see below

Standard Variables

- Some variables used by convention in shells:
  - $IFS specifies the inter field separator
  - $HOME specifies the users home directory
  - $USER or $LOGNAME
  - $SHELL specifies the shell being run
  - $TERM specifies the terminal type
  - $PS1 and $PS2 specifies the prompts for the shell
  - $PATH specifies in what order to search directories
  - $MANPATH specifies search directories for man pages

Built-in Variables

- The built in variables are of great import for scripts
  - $? Has the exit status of the last process
  - $$ has the process ID number of the current shell
  - $! Has the process ID of the last background process
  - $- has the flags passed to the shell when invoked
  - $# has the number of arguments passed to the shell
  - $* has all the arguments
  - $@ is the same except
    - "$@" allows arguments that were quoted to be replicated

Directory related variables

- ~ = home directory
- ~name = home directory of name
- ~+ = current working directory
- ~- = previous working directory

Korn Shell Variable Control

- The Korn shell offers variable checking:
  - ${#var} specifies the length of var
  - ${#*} specifies the number of command line arguments
- The shell also offers control
  - ${var:Xvalue}
    - X - value is expanded and used if var is not set or null
    - X = same as– but var is set to value
    - X ? If `var` is null or unset value is displayed and the script exits
  - ${varYpattern}
    - Y # removes minimal matching pattern prefix; (##) removes max
    - Y % removes min matching pattern suffix; (%%) removes max

Simple manipulations(1)

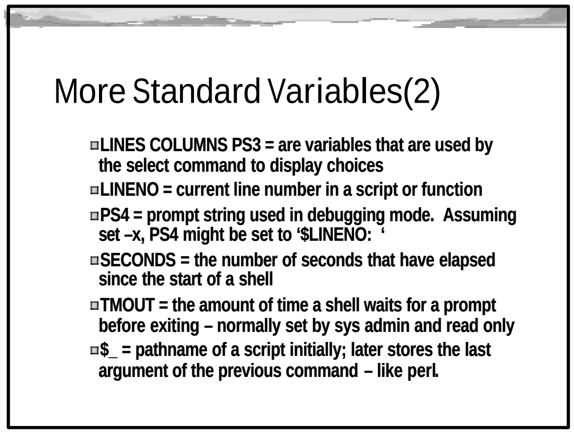
- **PATH=/xyz/bin/:$HOME/bin/**
- **PATH=$PATH:/xyz/abc/def/**
  - **This sets the path to the old path plus a new directory**
- **PATH=/xyz/abc/def/:$PATH**
  - **Puts the current directory at the front of the search list**

Simple manipulations(2)

- **PS1= some different prompt for the shell – below is all the junk I might imagine:**
  - **PS1=$HOSTNAME:$LOGNAME:$PWD:!\>**
    - **! is the current command number**
    - **\> escapes the > so it is taken as a literal**
  - **Examine the impact of quoting**
    - **PS1=${PWD##/*/}:!\> – fixed at time var set**
    - **PS1="${PWD##/*/}:!\>" – fixed at time var set**
    - **PS1='${PWD##/*/}:!\>' – interpreted when "run"**

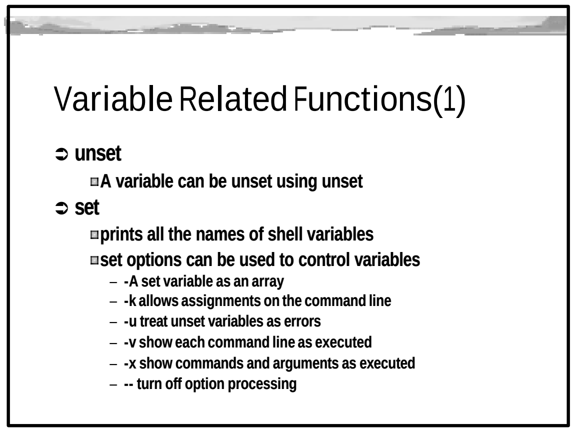
More Standard Variables(1)

- **The Korn Shell has about a dozen standard variables it sets. The most interesting are:**
  - **ENV = the name of a startup file**
  - **PWD and OLDPWD = the current previous working dir**
  - **PPID = process number of the shells parent**
  - **FPATH = the path to search for function files**
  - **RANDOM = provides a random number**
  - **HISTFILE and HISTSIZE = the name of the command history file and the number of commands kept**



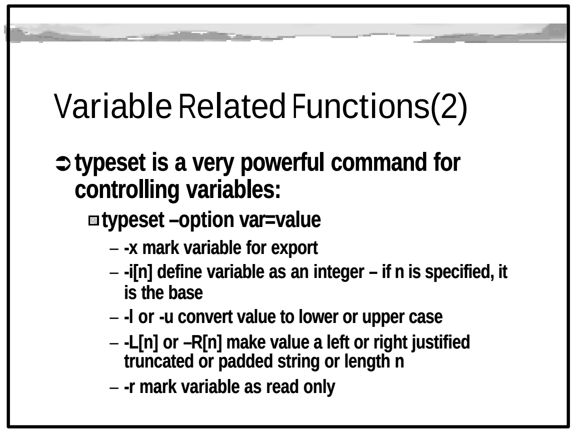
More Standard Variables(2)

- LINES COLUMNS PS3 = are variables that are used by the select command to display choices
- LINENO = current line number in a script or function
- PS4 = prompt string used in debugging mode. Assuming set –x, PS4 might be set to '$LINENO: '
- SECONDS = the number of seconds that have elapsed since the start of a shell
- TMOUT = the amount of time a shell waits for a prompt before exiting – normally set by sys admin and read only
- $_ = pathname of a script initially; later stores the last argument of the previous command – like perl.



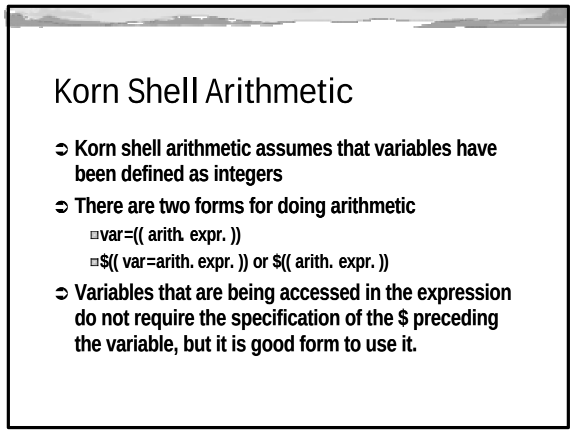
Variable Related Functions(1)

- unset
  - A variable can be unset using unset
- set
  - prints all the names of shell variables
  - set options can be used to control variables
    - -A set variable as an array
    - -k allows assignments on the command line
    - -u treat unset variables as errors
    - -v show each command line as executed
    - -x show commands and arguments as executed
    - -- turn off option processing



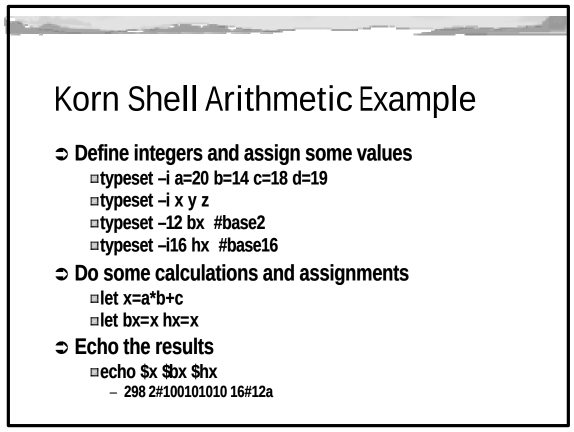
Variable Related Functions(2)

- typeset is a very powerful command for controlling variables:
  - typeset –option var=value
    - -x mark variable for export
    - -i[n] define variable as an integer – if n is specified, it is the base
    - -l or -u convert value to lower or upper case
    - -L[n] or –R[n] make value a left or right justified truncated or padded string or length n
    - -r mark variable as read only



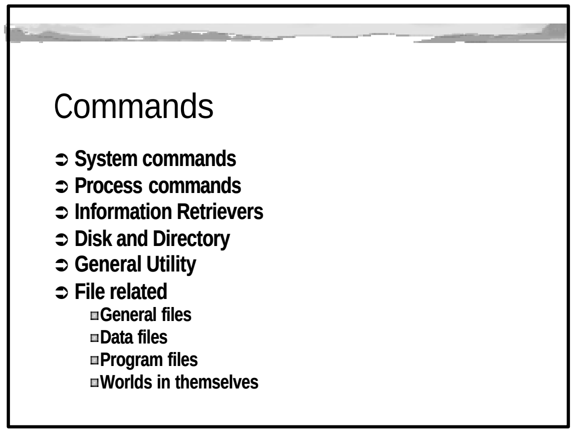
Korn Shell Arithmetic

- Korn shell arithmetic assumes that variables have been defined as integers
- There are two forms for doing arithmetic
  - var=((arith. expr.))
  - $((var=arith. expr.)) or $((arith. expr.))
- Variables that are being accessed in the expression do not require the specification of the $ preceding the variable, but it is good form to use it.



Korn Shell Arithmetic Example

- Define integers and assign some values
  - typeset –i a=20 b=14 c=18 d=19
  - typeset –i x y z
  - typeset –12 bx #base2
  - typeset –i16 hx #base16
- Do some calculations and assignments
  - let x=a*b+c
  - let bx=x hx=x
- Echo the results
  - echo $x $bx $hx
    - 298 2#100101010 16#12a



Commands

- System commands
- Process commands
- Information Retrievers
- Disk and Directory
- General Utility
- File related
  - General files
  - Data files
  - Program files
  - Worlds in themselves

System Commands(1)

- echo – allows status information or debugging
  - ksh echo does not allow –n, printf preferred
- passwd – allows you to change your password
- chgrp – change the group to which a file belongs
- chmod – change the protections on a file
- clear – clear the display
- stty – set terminal I/O properties
- touch – change the dates of last access for a file – if the file named doesn't exist, it will be created

System Commands(2)

- set – listing of variables
  - option switches allow control of how variables are set
- unset – makes a variable undefined
- typeset – allows control of the values assigned to variables
- xargs – a mechanism for allowing more than ten arguments to be passed to a command
- tee – duplicate standard input sending one copy to a named file and another copy to standard output

Process Commands(1)

- bg – places a suspended process (^Z) in the background. fg moves the last background process to the foreground.
- nice – runs a command (with arguments) at a lower priority
- ps – lists processes
- sleep – wait a specified number of seconds before execuitng another command
- kill – stop a process

Process Commands(2)

- at, atq, atrm – at runs a command at a specified time. atq check the queue and atrm removes a given scheduled job.
- nohup – allows a command to be run separated from the parent process such that the command continues to run after the user logs out.
- time – run a command showing time used. (timex – also runs a command, but allows more options)
- truss – show system calls and signals for a provided command or a process id.

Information Retrievers(1)

- date – prints the current date and time
- finger – displays data about one or more users
- groups – show the groups a user belongs to
- id – list user and ids – individual and group
- logname – lists your login name
- env – displays the current environment variables – similar to set without options
- hostname – prints the name of this host

Information Retrievers(2)

- type – describe the type of a command – I.e built in, function, external,
- which – list the fully qualified pathname of a command
- apropos – lookup keywords for man pages and display the man pages that may be relevant
- man – display a man page
- whatis – print a brief description of a program

Information Retrievers(3)

- ➲ w – print systems status and who is on
- ➲ who – print current sessions
- ➲ users – list logged in users in a space separated list – like who
- ➲ fgrep – simple file search program – doesn't use patterns
- ➲ grep – general regular expression program to find patterns in text (egrep extended version)

Simple Directory Commands

- ➲ cd – change to a named directory
- ➲ pwd – print the current working directory
- ➲ ls – list information about a file

Disk and Directory

- ➲ mkdir – create a directory
- ➲ rmdir – remove a directory
- ➲ df – show free disk blocks for all mounted drives
- ➲ du – show disk usage for the named directory
- ➲ find – find a file in a directory subtree
  - ◻Need to specify the name being searched for
  - ◻Need to specify print to rpint the name when found
  - ◻Was designed to execute commands on found files
- ➲ dircmp – compare the contents of two directories

General Utility

- cal – a utility to print a calendar
- calendar – an appointment management system
- dc – an interactive desk calculator
- bc – a program to do arbitrary precision arithmetic in multiple bases
- od – produces a dump of a file – an octal dump. Many switches allow additional forms of display.

File Related Commands (Common)

- cat – list file contents to the screen; it can be used to join a set of files together
- cp – copy a file
- diff – compare two files for differences
- mv – move or rename a file
- rm – remove a file
- ln – with the –s option, create a symbolic link to a file. With –s, deleting the link does not delete the source. Without –s the link is the same as the file

File Related Commands (2)

- tr – subsitute chars in string2 for chars in string1
- head – look at the starting lines of a file
- tail – look at the ending lines of a file
- file – provides information about the types of files
- fgrep – simple form of grep and egrep for finding none regular expression patterns
- fmt – fills and joins text – simple formatting
- pr – a simple formatting program for files
- wc – count the characters, words, and lines in a file

Data File Related Commands

- cut – cut columns out of a file
- dd – copy and convert the input file to an output file doing a number of conversions
- join – join columns of two files based on common ids
- paste – join files into a common file of multiple columns
- sort – sort a file based on contents
- uniq – remove adjacent duplicate lines – often used with sort
- split – splits a file into files of a given number of lines
- csplit – splits a file based on a pattern

Commands related to transfer

- compress – one of a family of programs to compress a file using Lemple-Ziv. Some systems will have zip, gzip, or other compression programs
- uncompress – the companion program to compress
- tar – move files in and out of a "tape" archive
  - Options are –c create –u update –x extract
  - -f followed by filename provides the target
- ar – move object files in and out of a library archive
- zcat – like uncompress except that it puts the file to standard out

Programming File Related

- nl – number the lines in a file
- strings – search binary files for string of more than four characters
- expand – expand tab characters into spaces
- unexpand – convert multiple spaces into tabs
- uuencode – allows a file with binary characters to be encoded such that it can be mailed without problems
- uudecode – the companion to uuencode

Functions and Aliases

- ➲ **aliases**
  - – alias str ="command"
  - – eg. alias dir="ls -al | grep '^d'
  - – alias -x exports the alias to sub shells
- ➲ **functions**
  - – function name {
  - – definition
  - – }
  - ▫ use “export name” to make a function available to spawned processes
  - ▫ Functions can manipulate command line arguments
  - ▫ In scripts, function arguments hide command line arguments

Some Simple Uses of alias

- ➲ **alias type=cat**
- ➲ **alias –x dir=“ls –l”**
- ➲ **alias –x pdir=“ls –l | more”**
- ➲ **alias sp=‘echo $PATH | tr “:” “\n” | sort’**
- ➲ **alias wd=“cd /home/spring/projects/current”**

A Simple Function

```
function sys
{
   printf "The time is: ";
   w | head -1 | cut -c0 -8;
   printf "System stats: ";
   w | head -1 | cut -c9 -70;
   printf "Number of user shells: ";
   echo $(( `w | wc -l` - 2 ));
   printf "Number of processes: ";
   echo $(( ` ps -ef | wc -l` - 1 ));
   printf "Number of different process owners: ";
   echo $(( ` ps -ef | cut -c0 -9 | sort | uniq | wc -l` -1 ));
   printf "Number of root processes: ";
   ps -ef | cut -c 0-9 | grep root | wc -l;
}
```

Shell history and editing

- Use set –o vi to set the editing mode to vi
  - This should be done in .profile
  - Use "ESC" to invoke the editor
  - Use j and k to move up and down the sequence
  - Use history to reissue commands
  - Consider installing the bash shell for even easier command line editing

Details on selected Commands

- ls
- sort and uniq
- vi
- man
- expr
- grep, egrep, and fgrep
- dd
- test
- find

ls

- Some of the ls options
  - -a will list both .files as well as all others
  - -l will provide all file information
  - -R will recursively list subdirectories
  - -t, -u list files by modification or access time
- Some games we might want to play
  - ls | wc –l – count the files
  - cat `ls *.txt` | more – page through all the text files

sort and uniq

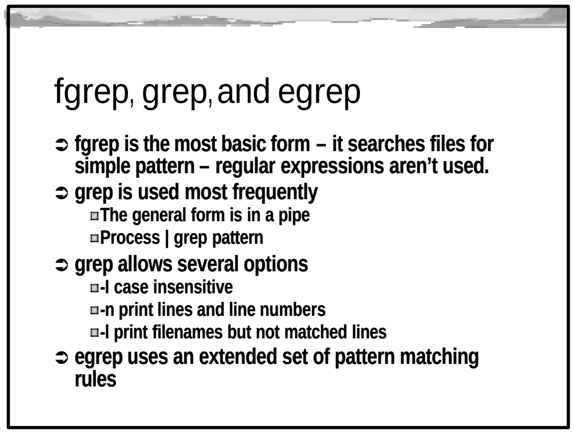
- Some of the sort options
  - -b ignore leading spaces
  - -d sort in dictionary order, ignoring punctuation
  - -f ignore case
  - -r reverse the sort order
  - -tc field separator is the character c
  - -n skip n fields before starting sort
- Some of the uniq options
  - -n ignore first n fields
  - -c print lines once with count
- Some games
  - sort records.dat | uniq

man

- man is generally used by simply typing man topic and prints the man page on topic
- man –k keyword prints a one line summary of any command that has a keyword matching keyword
- man –s section topic prints the man page for topic found in section
  - section 1 is user commands
  - section 2 is system call
  - section 3 is functions, etc.

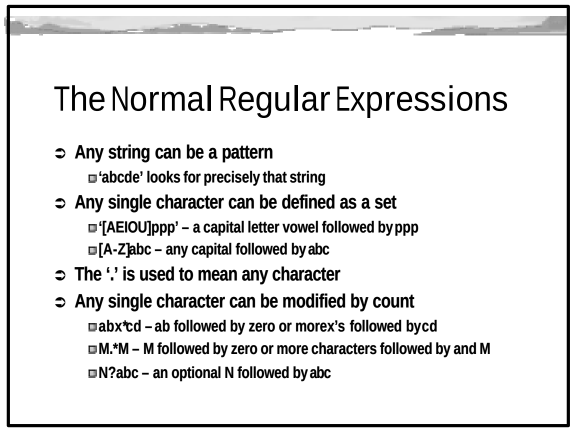
Regular expressions

- In Unix, patterns can be used to match strings. These patters are called regular expressions
  - The shell uses simplified regular expressions for files (see above)
  - The Korn shell uses an expanded set of file expressions (see above)
  - grep uses a “more normal set” and egrep uses an expanded regular set



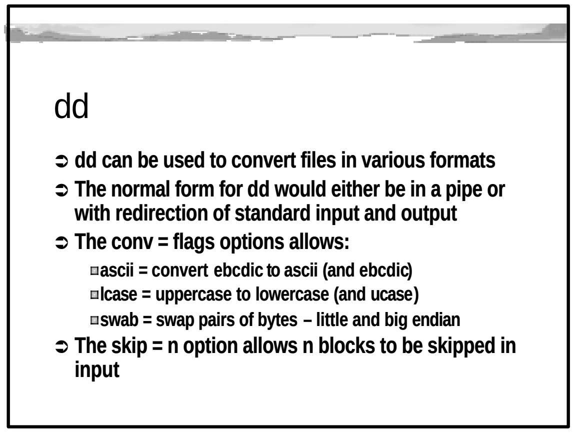
fgrep, grep, and egrep

- fgrep is the most basic form – it searches files for simple pattern – regular expressions aren't used.
- grep is used most frequently
  - The general form is in a pipe
  - Process | grep pattern
- grep allows several options
  - -I case insensitive
  - -n print lines and line numbers
  - -l print filenames but not matched lines
- egrep uses an extended set of pattern matching rules



The Normal Regular Expressions

- Any string can be a pattern
  - 'abcde' looks for precisely that string
- Any single character can be defined as a set
  - '[AEIOU]ppp' – a capital letter vowel followed by ppp
  - [A-Z]abc – any capital followed by abc
- The '.' is used to mean any character
- Any single character can be modified by count
  - abx*cd – ab followed by zero or more x's followed by cd
  - M.*M – M followed by zero or more characters followed by and M
  - N?abc – an optional N followed by abc



dd

- dd can be used to convert files in various formats
- The normal form for dd would either be in a pipe or with redirection of standard input and output
- The conv = flags options allows:
  - ascii = convert ebcdic to ascii (and ebcdic)
  - lcase = uppercase to lowercase (and ucase)
  - swab = swap pairs of bytes – little and big endian
- The skip = n option allows n blocks to be skipped in input

find

- The find command is often used to locate a file. It searches subdirectories from a given starting point:
  - find ~spring –name xyz –print
- Searching the entire file system using wildcards:
  - find / -name *.c –print
- Commands can be executed for each find
  - find / -name core –exec rm –f {} \;
  - {} places the filename and \; indicates command end
  - find / -name core –ok rm –f {} \; causes interactive confirmation

Introduction to Scripts

- Scripts can do anything that can be done on the command line
- Scripts also have a set of loops and control structures
- Normally, the first line of a script is the location of the shell. The line takes the form:
  - #! /usr/bin/ksh – or whatever the path of the shell is
- Comments are preceded by a # and continue to the end of the line

A First Simple Script

```
echo "The number of arguments is $#"
echo "The argument string is \"$*\""
count=0;
for  i in $*
do
   count=`expr $count + 1`
   echo "Argument $count. $i"
done
```