
Intermediate UNIX

Science & Technology Support Group
High Performance Computing
Ohio Supercomputer Center
1224 Kinnear Road
Columbus, OH 43212

Table of Contents

- **Basic UNIX—Review**
- **Intermediate UNIX**
 - **History**
 - **Alias**
 - **File Name Completion**
 - **Shell Scripts**
 - **.login and .cshrc Files**
 - **File-Management Commands**
 - **Text Processing**
 - **Advanced vi Editing**
- **Selected References**

Purpose of Course

- Present elements of Intermediate UNIX
- Provide hands-on practice

Acknowledgments

- **Course notes are based on:**
 - UNIX Primer Plus, Third Edition**
Don Martin, Stephen Prata, Mitchell Waite, Michael Wessler, and Dan Wilson
Waite Group Press, A Division of Macmillan USA, Inc., 201 West 103rd Street, Indianapolis, Indiana, 46290 USA © 2000
- **Science and Technology Support Group, OSC**
 - Leslie Southern
 - Troy Baer
 - Scott Brozell
 - Pete Carswell
 - Dave Ennis
 - Jim Giuliani
 - Elaine Landwehr

Basic UNIX--Review

- **UNIX commands**
 - always end with a return
- **Commands you should already know**
 - `date`
 - `cal`
 - `man`
 - `who`
 - `finger`

Basic UNIX--Review

Further commands you should already know

- `ls`
- `cat`
- `more`
- `less`
- `pr`

Basic UNIX--Review

Further commands you should already know

- `mkdir`
- `cd`
- `pwd`
- `rmdir`
- `rm`
- `cp`
- `mv`
- `ln`

Basic UNIX--Review

Metacharacters

- *
- ?
- [*character*]

Directory Abbreviations

- .
- ..
- ~

Basic UNIX--Review

Redirection

- >
- >!
- >>
- <
- < > or > < (never use two inputs or two outputs in the same command)
- | (pipe)
- tee (a command)

Job Control

- **Ctrl-z**
 - stops a job that is in process
- **fg**
 - (foreground)
 - resumes a job where you left off
- **&**
 - *command* &
 - places a job in the background and returns the personal job control number and the process identification (PID) number
- **jobs**
 - lists and labels jobs on your terminal
 - job number
 - + = current job
 - - = next current job

Job Control, continued

- `ps`
 - reports on processes currently on the computer
- `kill`
 - use this command with job number or the PID number

Intermediate UNIX

- **History**
- **Alias**
- **File Name Completion**
- **Shell Scripts**
- **.login and .cshrc Files**
- **File-Management Commands**
- **Text Processing**
- **Advanced vi Editing**

History

- `cs`
- remembers 20 commands (number can be changed)
- able to repeat earlier commands
- able to modify earlier commands before repeating
- able to set it for current session or permanently
 - `set history=20`
 - modify `.cshrc` file

History—try this

- type some commands like `ls`, `ls` with options, `pwd`, `cal`, `date`, etc.
- type `history`
- to return an earlier command, use `!` and
 - `!#`
 - `!letter`: run most recent command from list that begins with *letter*
 - `!-#`: run command that is `#` from most recent command
 - `!pattern` (can include metacharacter): look for most recent command with *pattern*
 - `!!`: repeat previous command

History—adding to `history` command

- **history substitutions**
 - translating event identifiers into ordinary commands
 - can be incorporated into longer commands
 - history reference—not necessarily first part of new command line

Command-Line Editing--Basic

- **format**
 - `^incorrect.chars^correct.chars`
 - substitutions begin at first instance of pattern
- **for ksh**
 - usually `<ESC> k`
 - takes you through history of commands
 - puts you in editor mode of `vi`
 - press `<ESC> k` repeatedly until desired command appears, then edit it

Word Identifiers

- **to use just part of an earlier command**
 - words numbered left to right
 - first word (command): 0 (zero), followed by a colon
 - words after command are 1-n
 - shortcuts—can be used without the colon preceding them
 - ^ word number 1
 - \$ last word
 - * all words after word 0
- **not used alone—appended to event identifier**
 - e.g., !3:2
 - third command from most recent, second word (not counting command)
 - can use a range, e.g. 2-4

History--Summary

Name	Options	Arguments	Notes
<code>history</code>	<code>none</code>	<code>none</code>	lists last <i>n</i> commands run by <code>cs</code> <code>set history=<i>n</i></code>

Identifying commands or events:

`!n` command number *n* from history list

`! -n` *n*th command before present one

`!c` most recent command beginning with character *c*

`!?pat?` most recent command containing pattern *pat*

History—Summary, continued

Identifying words within an event:

0	command (first actual word)
: <i>n</i>	<i>n</i> th word in an event
:^ or ^	same as :1; (with ^, colon not necessary)
:\$ or \$	last word in event ; (with \$, colon not necessary)
: <i>n1-n2</i>	words <i>n1</i> through <i>n2</i>
^-\$	words 1 through last word
*	words 1 through last, or no word if no word 1

History—Summary, continued

Special notes

- **!!** preceding command line
- **!** preceding command line (needs word identifier)
- **!*** options, arguments of preceding command

Alias

- **purpose**
 - rename, redefine, rearrange commands
- **can be temporary (for the session only) or permanent (put in .cshrc or .profile)**
- **can use arguments**
- **can be used for compound commands**
 - more than one command, each separated by semicolon
- **can be used for complex commands**
 - use \!* to stand for arguments of preceding command
 - \ strips history significance from !
- **can use aliases in aliases**
 - aliases need not be commands: can be programs or procedures you've written

Alias--Summary

Name	Options	Arguments	Notes
<code>alias</code>	<code>none</code>		prints list of aliases
<code>alias</code>		<i>abb command</i>	creates alias
<code>unalias</code>		<i>abb</i>	removes alias

File Name Completion

- **purpose**
 - for `cs`: to speed up typing
- **format**
 - command `chars<ESC>`
 - if more than one file has common characters, result will show the next common letters, then you type another character followed by `<ESC>`
- **temporary**
 - at command line, type: `set filec`
- **permanent**
 - put `set filec` in `.cshrc` file

Shell Scripts

- **purpose**
 - to construct new commands by creating a file with commands
 - when you run the shell script, new shell opens for duration of command, then closes
- **method**
 - create a shell script file with the command(s) you want to use
 - `cs`h *shell.script* or `sh` *shell.script* OR
 - `ch`mod `u+x` *shell.script*
 - then at prompt, type *shell.script*
- **flexible**
 - script can contain more than one command: separate with semicolons
 - command can use command-line arguments
 - can create loops and other programming features

Shell Scripts—Arguments

- **purpose**
 - to allow shell script to keep track of arguments you type after script name
- **format**
 - use `echo` plus `$n`
 - numbering begins with `$0`
- **sample**

Shell Variables

- **purpose**
 - variables are names that can have different values assigned
 - some are built-in
 - some you define

Shell Variables—Built-in

- **cs**
 - **two kinds**
 - **shell variable**
 - **known just to shell in which it's created**
 - **environment shell variable**
 - **known to shell in which defined AND descendant shells**
- **to determine ordinary shell variables your shell knows**
 - **set**
 - **lower case**
- **to determine environment variables**
 - **setenv**
 - **capitalized**

Shell Variables—Obtaining Value

- **echo**
 - use `$` to get value of the variable
 - can use `$` construction in other expressions
 - e.g., to make a shell script more general, thus more versatile
 - `cp $1 $HOME`
 - anyone can use this, and file will be copied into that person's `$HOME`

Shell Variables—Setting

- `set var.name = value`
 - check to see if it were created by typing `set`
 - get rid of this variable by typing `unset var.name`
- `setenv VAR.NAME value`
 - no equal sign
 - variable name is capitalized

.login and .cshrc Files

- **hierarchy for variables**
 - shell created by UNIX when you log in (home, PATH)
 - **.cshrc**
 - treated as shell script
 - executed for each new `cs` shell
 - get a standard version from sys admin
 - used to set shell variables
 - preferred for aliases
 - ordinary text file
 - **.login**
 - treated as shell script
 - executed only when you log in
 - get a standard version from sys admin
 - used to set shell variables
 - preferred for setting environment variables—to carry over to later shells
 - ordinary text file

Shell Metacharacters

Function	Metacharacter
wildcard substitution	* ? []
redirection	> >> <
background process	&
command separator	;
continue command on next line	\
value of a variable	\$
history prefix	!
get value of command	` `

Neutralizing Shell Metacharacters

Function	Character
negate special meaning of next character (even negates itself)	\
negate special meaning of characters within them	''
negate special meaning of characters within them, except \$, `, and \	""

also '' combines several words into one argument

File-Management Commands

Name	Options	Arguments	Purpose
<code>wc</code>	<code>-lwc</code> lines, words, characs	<code>[filename(s)]</code>	counting program; if more than one file is given, gives counts for each file, then the totals (can save output to new file)
<code>tail</code>	<code>[+/-n][lbc]</code> + starts <i>n</i> from beginning - starts <i>n</i> from end lbc lines, blocks, characs	<code>filename(s)</code>	shows end of named file; default—last 10 lines
<code>head</code>	<code>[-n]</code>	<code>filename(s)</code>	shows first <i>n</i> lines of named file; default—10

File-Management Commands, cont'd

Name	Options	Arguments	Purpose
sort	<ul style="list-style-type: none">-b ignore leading tabs, spaces-d dictionary order: letters, digits blanks determine order-f ignore case-n sort numbers by arithmetic value-o <i>filename</i> place output in <i>filename</i> instead of to standard output-r sort in reverse order-u suppress duplicate lines	<i>filename(s)</i>	sorts and merges lines from file(s)

File-Management Commands, cont'd

Name	Options	Arguments	Purpose
<code>uniq</code>	<code>-u</code> prints non-duplicated lines <code>-d</code> prints only duplicated lines <code>-c</code> prints number of times line appears in file, then the line	<i>input</i> [<i>output</i>]	prints unique lines and single copy of duplicated lines (if no option given)

File-Management Commands, cont'd

Name	Options	Arguments	Purpose
lpr	vary	[<i>filename</i>]	sends file to printer queue
lpq			reveals printer queue and identifying information
lprm		<i>file ident</i> can be <i>ownername</i> <i>ident-number</i> <i>filename</i>	removes file from printer queue

File-Management Commands, cont'd

Name	Options	Arguments	Purpose
<code>time</code>	none	<i>command</i>	runs the command and returns a summary of time used

`calendar` not a command that you execute; reminder system; create a file entitled `calendar`: each item should have a valid date (abbrevs, all numbers, full words accepted—use month-day order) and an item for reminding; you should receive a reminder on the day before and on the day

Text Processing

- **creating text files**
 - e.g., with `vi`
- **modifying text files**
 - e.g., with `spell`
 - editing with `sed`
- **rearranging text files**
 - e.g., with `join`
- **formatting text files**
 - e.g., with `nroff`

Filter

- **steps**
 - input from standard input
 - process input
 - output to standard output
- **can use UNIX redirection and pipes**
 - to allow input from file or command and to send output to file or another command
- **can use echo in shell scripts for descriptive phrases**

join

- **combines files**
- **contrast cat command**
 - combines two files by appending one to the other
- **join combines files by adding lines of file 1 to end of corresponding lines of file 2**
- **steps**
 - sort lines of file (ascending ASCII)
 - ensure lines to be joined have same join field (normally blank or tab that divides line into fields)

join--Summary

Name	Options	Arguments
join	-tc use c as field delimiter -jn m use field m of file n as the join field	<i>file1 file2</i>

sed—Stream Editor

- **wholesale editing work**
- **how it works**
 - **takes a file as input**
 - **reads, modifies one line at a time**
 - **sends output to the screen**
- **efficient for large files**
- **can be used with pipes and redirection**
- **often used in shell scripts**

sed—Basics

- **pattern**
 - `sed 'ed-command' filename`
- **process**
 - looks at each line
 - applies command, if appropriate
 - prints result
 - in basic use, leaves original file unchanged
- **use for global changes**
 - a little too much trouble for just a couple of changes
- **can redirect output into file**
 - again, original file unchanged
 - new file contains changes

sed—Editing Instructions

- **instruction—two parts**
 - **address specification**
 - which lines are affected
 - if no address given—all lines are affected
 - can be specified numerically or by pattern matching
 - **command**
 - what to do to the affected lines

sed—Common Commands

Command	Action
<code>a \</code>	appends following line(s) to affected lines
<code>c \</code>	changes affected lines to following line(s)
<code>d</code>	deletes affected lines
<code>g</code>	makes substitutions affect every matching pattern
<code>i \</code>	inserts following line(s) above affected lines
<code>p</code>	prints line, even under <code>-n</code> option
<code>q</code>	quits when specified line is reached
<code>r <i>filename</i></code>	reads <i>filename</i> ; appends contents to output
<code>s/<i>old</i>/<i>new</i>/</code>	substitutes <i>new</i> for <i>old</i>
<code>w <i>filename</i></code>	copies line to <i>filename</i>
<code>=</code>	prints line number
<code>! <i>command</i></code>	applies <i>command</i> if line is <i>not</i> selected

sed—Specifying Lines

- **two methods**
 - **specify address with a number**
 - single number to indicate a particular line
 - e.g., `sed `3d` filea`: deletes (d) third (3) line from file entitled `filea`
 - range of lines: two numbers separated by comma
 - e.g., `sed `2,4 s/z/#/` fileb`: in lines 2-4, substitutes the first “z” in each line with “#”
 - **specify a pattern (place pattern between slashes)**
 - e.g., `sed -n `/zebra/p` filec`: prints (p) only those lines (-n) that contain the pattern `zebra` in `filec`
 - **can combine the two methods**
 - e.g., `sed `1,/aardvark/d` filed`: deletes (d) line 1 (1) through the first line that contains the indicated pattern (`aardvark`) in `filed`

sed—a \, c \, i \

- **format**

- sed 'a\
pattern' *filename*
- sed 'c\
pattern' *filename*
- sed 'i\
pattern' *filename*

sed—Pattern Matching

Metacharacter	Action
\	turns off special meaning of next charac
^	matches beginning of line
\$	matches end of line
.	matches single charac
[]	matches any one of enclosed characs; characs can be listed individually or as range (with -)
<i>pat*</i>	matches zero or more occurrences of <i>pat</i>
&	used in <i>newpattern</i> part of an <i>s</i> command to represent reproducing <i>oldpattern</i> part

sed—Multiple Commands

- **method**
 - add in each command, one command to a line
 - enclose entire command list in single quotes
 - backslash at end of command means continue command to next line
- **multiple commands are processed in order given**

sed—Tags

- **allow you to designate part of an expression**
 - compare to `&`, which allows you to refer to an entire expression

sed—Shell Scripts

- can place `sed` command configuration in shell script
- make shell script file executable (`chmod`)
- format
 - *shell.script filename redirection.if.desired*
 - redirection can be part of shell script, e.g., print, save to a file

nroff—Text Formatting

- purpose
 - set up margins
 - set up line spacing
 - format paragraph
 - etc.
- levels of simplicity and power
 - use on ordinary file
 - use `nroff` macro
 - put `nroff` commands in a file

nroff—Ordinary File

- **nroff *filename***
 - text runs off screen—default is 66 lines, and screen shows about 22 lines
- **nroff *filename* | more**
 - shows formatted text on screen
 - original file not altered
 - output can be redirected
- **options placed in files**
 - **.cc #** dot, two characters, if needed—space and number

nroff—Basic Formatting Requests

Request	Action
<code>.ce <i>N</i></code>	centers next <i>N</i> lines
<code>.in <i>N</i></code>	indents text <i>N</i> spaces
<code>.hy <i>N</i></code>	turns on auto-hyphenation, <i>N</i> \geq 1; off <i>N</i> =0
<code>.ll <i>N</i></code>	line length = <i>N</i> characs
<code>.ls <i>N</i></code>	line spacing to <i>N</i> (2=double space, etc.)
<code>.na</code>	doesn't right justify
<code>.nf</code>	doesn't fill text
<code>.sp <i>N</i></code>	puts in <i>N</i> blanks
<code>.ti <i>N</i></code>	next line indented <i>N</i> spaces (temporary indent)
<code>.tr <i>abcd..</i></code>	replaces a with b, c with d, etc.

nroff—mm Macro

- **macro**
 - command or instruction made from several more basic units
- **chief tool**
 - `.de CC`
 - ends with `..`
- **macro package**
 - set of predefined macros—handle common situations
- **usage**
 - insert macro requests in file
 - `nroff -ms filename`

nroff—Useful ms Macros

Macro	Meaning and Typical Usage
.TL	use next line for title
.AU	use next line(s) to specify author(s)
.AI	use next line(s) to specify institutions of author(s)
.AB	Abstract begins next line
.AE	comes after end of Abstract
.PP	begin paragraph with first line indented
.LP	begin a block paragraph aligned left
.IP <i>L</i>	begin indented block paragraph; <i>L</i> : labeled with charac <i>L</i>
.FS <i>L</i>	start footnote; <i>L</i> means footnote labeled with charac <i>L</i>
.FE	end footnote

nroff—tbl and eqn

- **tbl**
 - **ts** **table start**
 - **te** **table end**
 - **place table material between beginning and ending markers, including table instructions and data**
- **eqn**
 - **similar usage**

grep

- **format**
 - `grep pattern file(s)`
- **purpose**
 - search contents of each file named for pattern (word, name, phrase, etc.)
- **features**
 - print lines that contain the pattern
 - can use wildcards
 - can be used with other commands
 - output can be redirected

grep

- **format**
 - `grep [options] pattern [file(s)]`
- **purpose**
 - search contents of each file named for pattern (word, name, phrase, etc.)
- **features**
 - print lines that contain the pattern
 - can use wildcards
 - can be used with other commands
 - output can be redirected

grep—Forms of Regular Expression

- dot (.) any one character
- [] any one character in the bracket; can include range
- [^] any character other than those in the bracket
- ^ line beginning with the expression
- \$ line ending with the expression
- \c (c=any charac except digit or paren) matches c and turns off any special meaning

grep—Summary

Name	Options	Arguments
grep	-n precedes each matching line with its line number -c prints only a count of matching lines (does not print lines) -v prints only lines that do _not_ match pattern -w finds only complete words -i ignores case	<i>pattern filename(s)</i>

find

- **format**
 - `find directory_pathname search_criterion action`
- **purpose**
 - find files that satisfy some criterion and then act on them
- **features**
 - recursive search: descends through directories and subdirectories—
a branching search

find–Search Criteria

- **form**
 - identifying flag word, followed by a space and a word or a number
- **common search criteria (primaries)**
 - finding a file by name
 - flag: `-name word_or_charac`
 - can use wildcards, surrounded by single quotation marks
 - finding a file by last access
 - flag: `-atime #_days_since_last_access`
 - can use `+` before `#` to mean more than that many days ago
 - can use `-` before `#` to mean less than that many days ago
 - no `+` or `-` means exactly that many days ago
 - finding a file by last modification
 - flag: `-mtime #_days_since_last_modification`
 - can use `+` or `-` to indicate greater than or less than
 - finding files modified more recently than a given file
 - flag: `-newer filename`
 - finding file based on who owns file
 - flag: `-user userid`

find-Actions

- **three flag options**
 - **-print**
 - prints pathname for every file that matches the criterion
 - **-exec**
 - allows you to give a command to be applied to the found files
 - command follows flag and ends with a space, backslash, semicolon
 - { } can represent the name of the files found
 - **-ok**
 - acts like -exec except prompts you to okay the command on each file before it executes

find—More Complex Forms

- **ability to expand basic parts of find**
 - **directory pathname**
 - can give a list of directories to be searched
 - **primaries (search criteria)**
 - **!** negates a primary; precedes primary; isolated by space on either side
 - **list of two or more criteria in a row: finds those files that satisfy all criteria**
 - **separating two criteria with -o flag (isolated by space on either side): finds those files that satisfy one or the other criterion**

find–Summary

Name	Options	Arguments
find	none	<i>directory_pathname(s)</i> <i>search_criteria action(s)</i>
search criteria		
-name	<i>filename</i>	files named <i>filename</i>
-size	<i>n</i>	files of size <i>n</i> blocks
-links	<i>n</i>	files with <i>n</i> links
-atime	<i>n</i>	files accessed <i>n</i> days ago
-mtime	<i>n</i>	files modified <i>n</i> days ago
-newer	<i>filename</i>	files modified more recently than <i>filename</i>
actions		
-print		print pathnames of found files
-exec	<i>command</i> {} \;	execute command; {}=found file
-ok	<i>command</i> \;	execute command; confirm before execute

sort–Revisited

- **sort by fields**
 - **fields: can be defined**
 - default: empty, nonblank strings separated by blanks (space or tab)
 - separator can be something else, often a colon
 - **defining field separator**
 - use `-tc` option, with `c` desired character
 - **defining field to be basis of sort**
 - use `+n` to indicate number of fields to skip
 - use `-n` to indicate where to stop the comparison
 - **multiple fields**
 - will sort file based on first field pattern
 - then sorts each block of lines with identical fields by second field pattern
 - **subdividing fields**
 - `sort` will skip designated number of characters in a field before sorting
 - format: `sort +n.n filename`
 - **flag options and fields**
 - options applied globally or for certain fields: position of option letter
-

sort–Summary

Name	Options	Arguments
sort	-tc sets the field separator to be character <i>c</i> (blank is default) -b ignores leading blanks when comparing fields +n.m skips <i>n</i> fields and then <i>m</i> characters before beginning comparisons; +n is the same as +n.0 +n.m and -n.m stops comparison after skipping <i>n</i> fields from the beginning plus <i>m</i> characters; -n is the same as -n.0	<i>[filename(s)]</i>

Note: **b**, **d**, **f**, **n**, and **r** options: if appended to field locator flags and will apply only to that field; options appearing before field locators apply globally; if multiple fields given, sorting done by first field given, then by next field(s), then by whole line

awk–Basics

- **pattern scanning and processing language**
- **pattern scanning**
 - looks through a file for certain patterns
- **processing**
 - does something with the lines found
- **used as a file processor**
 - can produce a new file consisting of the original file's columns plus a new column that has performed some operation on any of the other columns
 - similar to a spreadsheet, at times, but can work with text as well as numbers

awk—Two Methods of Using

- `awk program filename`
 - `program` = instructions
 - `filename` = name of file to be acted on
- `awk -f file filename`
 - `file` = name of file containing program instructions
 - a bit more difficult
 - less likely to produce syntax error messages when you use symbols in program that also have special meaning to the shell

awk–Program

- consists of one or more program lines
- program line
 - consists of two parts
 - pattern
 - action enclosed in braces
 - pattern: simple string patterns are enclosed in slashes
- fields
 - defined as they are for `sort`
 - separated by blanks or some chosen character
 - choose other character with `-Fc` flag
 - labeling system
 - `$1`= first field, `$2`=second field
 - `$0`=entire line
 - can be used in patterns and actions both
- actions can be used with individual fields and with multiple fields
- arithmetic operators: `+`, `-`, `*`, `/`
- separate multiple actions with semicolon

Advanced vi Editing—Undo, Repeat

- **undo last command**
 - `u`
- **repeat last command**
 - `.` (dot)
- **moving cursor—not a command**
- **can combine certain commands with dot (repeat last command)**
 - e.g., if you don't want to do a global search/replace with `sed`, can do a search with `/`, make the change (e.g., `cw` for change word), press `n` to find next occurrence, press `.` (dot), and the change will be repeated (`n` does not constitute a command)

Advanced vi Editing—Abbreviations

- **format**
 - to set an abbreviation
 - `:ab abb word(s)`
 - to unset an abbreviation
 - `:una abb`
 - to see which abbreviations are set
 - `:ab`
- **used only in vi's text input mode**
 - just enter abbreviation followed by a space
- **for the session only**

Advanced vi Editing—map

- saves command sequences for repeated use
- shorthand in command mode
- format
 - `:map [x] [editing_commands]`
 - **x**: limited to specific single characters
 - symbols: + * \ =
 - characters: K V q g v
 - control keys: Ctrl-a, Ctrl-k, Ctrl-o, Ctrl-t, Ctrl-w, Ctrl-x
- method to create map macro
 - do editing once manually
 - write the keystroke sequence
 - enter the map command
- features
 - can be repeated with `.` (dot) command
 - can be undone with `u` command
 - for the session only

Advanced vi Editing—Multiple Files

- **advantages**
 - faster to start `vi` once, rather than several times
 - temporary creations (`map`, abbreviations) can be used for several files
 - yank and put lines from one file into a second or third file
- **traits**
 - multiple files are called sequentially
 - editor works on one file at a time
 - save changes in one file with `:w` command before going to next file
- **methods**
 - `vi` on one file then call next file with `:e` or `:r` command
 - with `:e` command, use `:e#` to switch to other file
 - always save before switching
 - list all files to be edited
 - `vi file1 file2 file3`
 - `vi file*` (use a wildcard)
 - `:n` calls up the next file (always save the earlier file)

Advanced vi Editing—Shell Commands

- **four ways to use shell while in editor**
 - run a shell command
 - temporarily escape to the shell
 - read in results of shell commands
 - filter text through a shell command
- **method**
 - most start with a colon and an exclamation point
 - command runs
 - return to vi editor
 - to run more than one command, create a shell
 - :sh
 - shell prompt appears
 - <Ctrl-d> to leave shell and return to editor
 - to read in results of a shell command
 - :r !*command*
 - :w (save)
 - can write parts of editing buffer to a shell command
 - :w !*command*
 - current editing buffer is the input

Advanced vi Editing—Search/Replace

- **two ways to search with vi**
 - */pattern* and *?pattern*
 - *:nm/pattern/*
- **general format of *:nm/pattern/***
 - *:address/command/parameter*
- **address part**
 - **two parts**
 - lines
 - patterns
- **metacharacters**
 - use special metacharacters to form search patterns: regular expressions

vi—Metacharacters

Character	Description
<code>\</code>	turns of special meaning of following charac
<code>^</code>	matches beginning of a line
<code>\$</code>	matches end of a line
<code>.</code>	matches any single charac except a newline
<code>*</code>	matches preceding charac or expression
<code>[string]</code>	matches any one of enclosed characs;
<code>[^string]</code>	matches any charac not enclosed
<code>&</code>	stands for the text in a search
<code>\(pat\)</code>	escaped parens used to define a pattern
<code>\n</code>	<i>n</i> refers to previous patterns defined by parens
<code>Ctrl-v</code>	escapes an Esc or Return in replacement part

Advanced vi Editing—Search/Replace

- two ways to search with vi
 - */pattern* and *?pattern*
 - *:nm/pattern/*
- general format of */pattern* and *?pattern*
 - *:address/command/parameter*
- can use regular expressions

Selected References

- **Arthur, Lowell Jay and Burns, Ted: UNIX Shell Programming, Wiley**
 - **Dyson, Peter, Kelly-Bootle, Stan, Heilborn, John: UNIX Complete, Sybex**
 - **Kochan, Stephen G. and wood, Patrick H., UNIX Shell Programming, Hayden Books**
 - **Lamb, Linda and Robbins, Arnold: Learning the vi Editor, O'Reilly**
 - **Martin, Don, Prata, Stephen, Waite, Mitchell, Wessler, Michael, Wilson, Dan: UNIX Primer Plus, Waite Group Press**
 - **Muster, John, UNIX Made Easy, Osborne**
 - **Ray, Deborah S. and Ray, Eric J.: Visual Quickstart Guide UNIX, Peachpit Press**
 - **Reichard, Kevin and Foster-Johnson, Eric: UNIX in Plain English, M&T Books**
 - **Taylor, Dave: SAMS Teach Yourself Unix in 24 Hours, SAMS**
 - **on-line information too numerous to mention**
-