A Guide to Unix Using Linux Fourth Edition

Presentation 4

Advanced File Processing

Objectives

- Use the pipe operator to redirect the output of one command to another command
- Use the *grep* command to search for a specified pattern in a file
- Use the *uniq* command to remove duplicate lines from a file
- Use the *comm* and *diff* commands to compare two files

Objectives (continued)

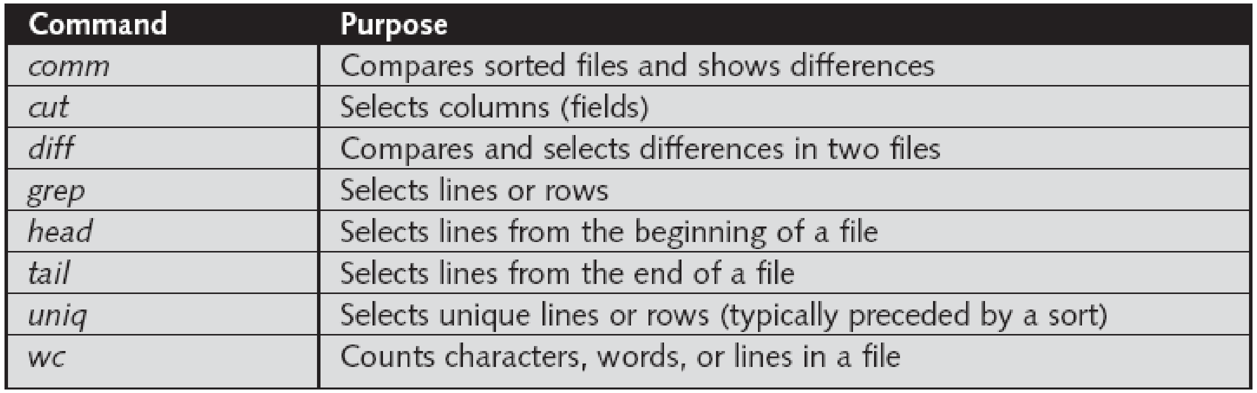
- Use the *wc* command to count words, characters, and lines in a file
- Use manipulation and transformation commands, which include *sed*, *tr*, and *pr*
- Design a new file-processing application by creating, testing, and running shell scripts

Advancing Your File-Processing Techniques

- Commands used for file processing can be organized into two categories:
  - **Selection commands**
    - Focus on extracting specific information from files
  - **Manipulation and transformation commands**
    - Alter and transform extracted information into useful and appealing formats

Advancing Your File-Processing Techniques (continued)

Table 5-1 Selection commands



Command	Purpose
comm	Compares sorted files and shows differences
cut	Selects columns (fields)
diff	Compares and selects differences in two files
grep	Selects lines or rows
head	Selects lines from the beginning of a file
tail	Selects lines from the end of a file
uniq	Selects unique lines or rows (typically preceded by a sort)
wc	Counts characters, words, or lines in a file

Advancing Your File Processing Techniques (continued)

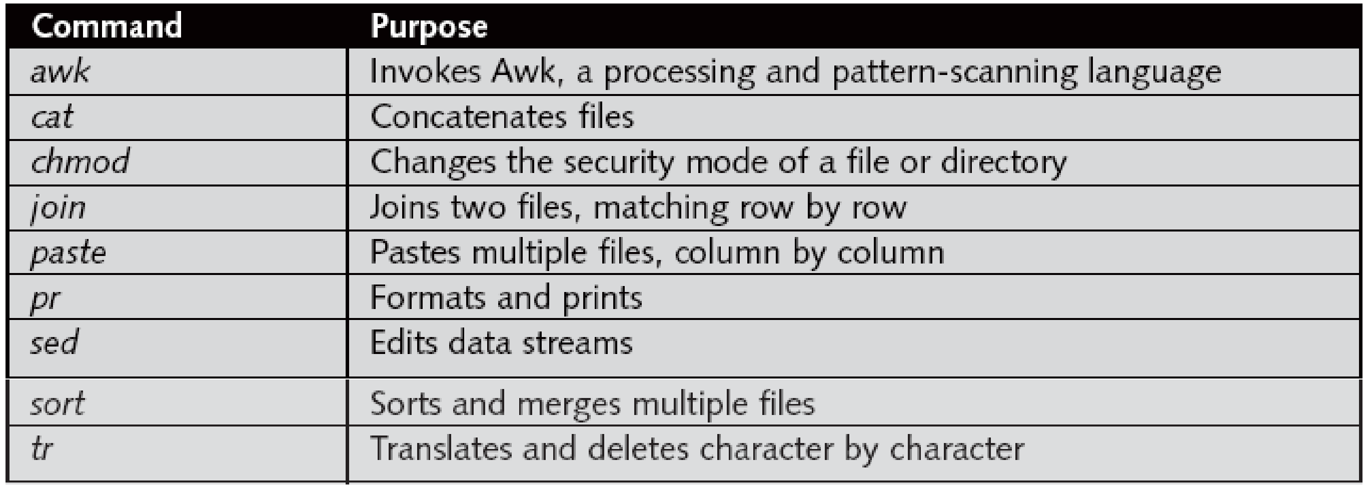


Table 5-2 Manipulation and transformation commands

Command	Purpose
awk	Invokes Awk, a processing and pattern-scanning language
cat	Concatenates files
chmod	Changes the security mode of a file or directory
join	Joins two files, matching row by row
paste	Pastes multiple files, column by column
pr	Formats and prints
sed	Edits data streams
sort	Sorts and merges multiple files
tr	Translates and deletes character by character

Using the Selection Commands

- The pipe (|) operator: another redirection operator
- Some useful selection commands:
  - *grep*
  - *diff*
  - *uniq*
  - *comm*
  - *wc*

Using the Pipe Operator

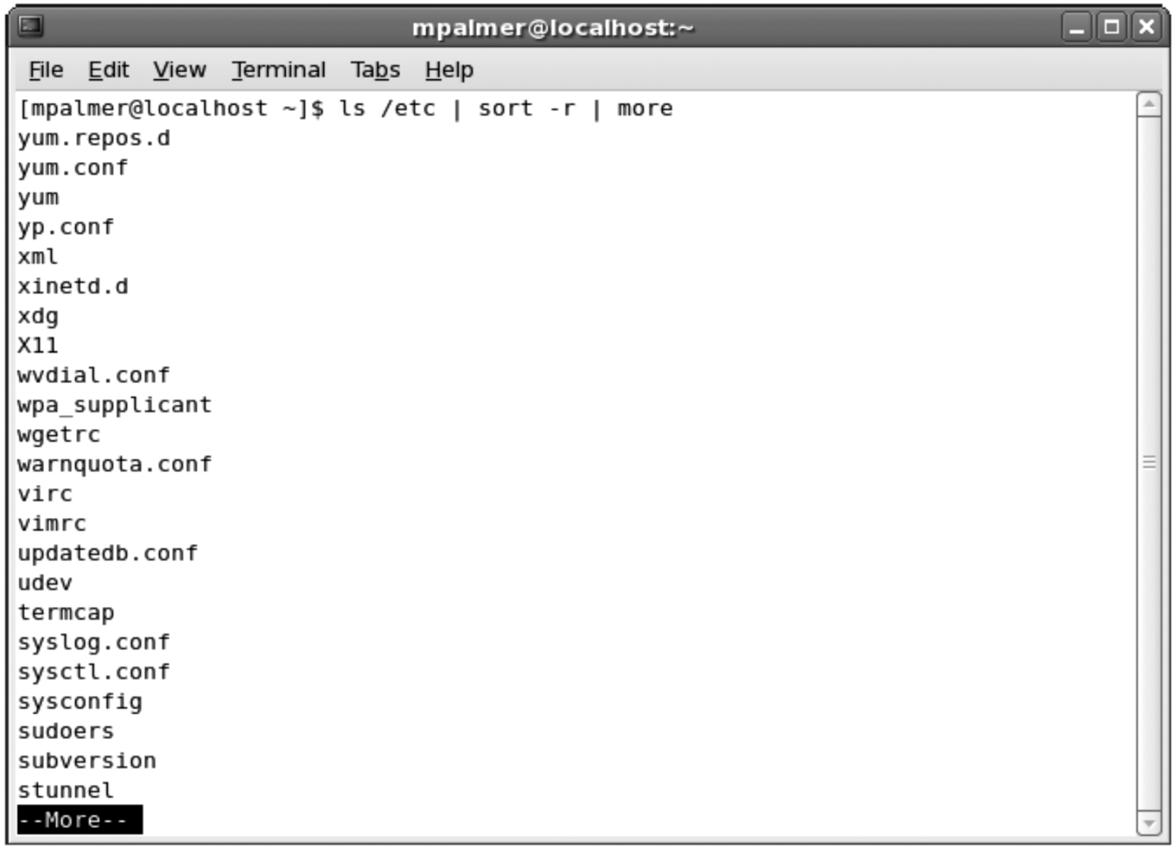
- < and > can be used for redirection
- **Pipe operator (|)** redirects the output of one command to the input of another command

```
first_command | second_command
```

- Pipe can connect several commands

```
first_command | second_command |
third_command ...
```

Using the Pipe Operator (continued)

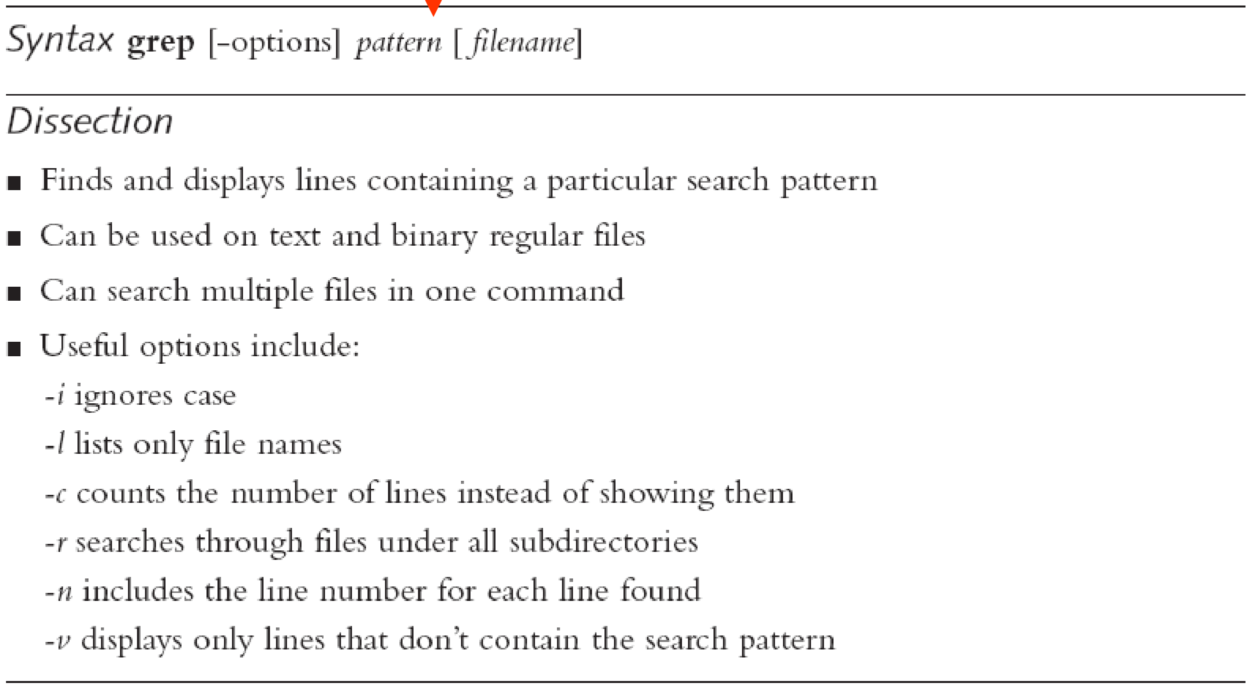


```
[mpalmer@localhost ~]$ ls /etc | sort -r | more
yum.repos.d
yum.conf
yum
yp.conf
xml
xinetd.d
xdg
X11
wvdial.conf
wpa_supplicant
wgetrc
warnquota.conf
virc
vimrc
updatedb.conf
udev
termcap
syslog.conf
sysctl.conf
sysconfig
sudoers
subversion
stunnel
--More--
```

Figure 5-1 Combining commands using the pipe operator

Using the grep Command

May be enclosed in single/double quotes



Syntax **grep** [-options] *pattern* [*filename*]

Dissection

- Finds and displays lines containing a particular search pattern
- Can be used on text and binary regular files
- Can search multiple files in one command
- Useful options include:

  -*i* ignores case

  -*l* lists only file names

  -*c* counts the number of lines instead of showing them

  -*r* searches through files under all subdirectories

  -*n* includes the line number for each line found

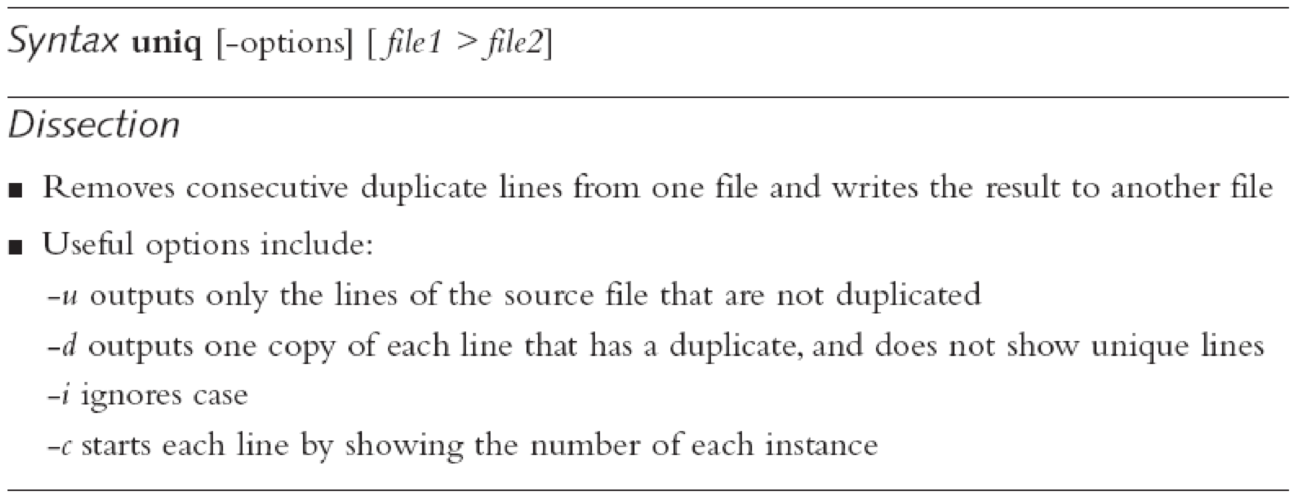
  -*v* displays only lines that don't contain the search pattern

- Example:

```
grep -r Computer Resources Committee /documentation
```

Using the uniq Command

- *uniq* removes duplicate lines from a file
- Compares only consecutive lines
  - Requires sorted input



Syntax **uniq** [-options] [*file1* > *file2*]

Dissection

- Removes consecutive duplicate lines from one file and writes the result to another file
- Useful options include:

  -*u* outputs only the lines of the source file that are not duplicated

  -*d* outputs one copy of each line that has a duplicate, and does not show unique lines

  -*i* ignores case

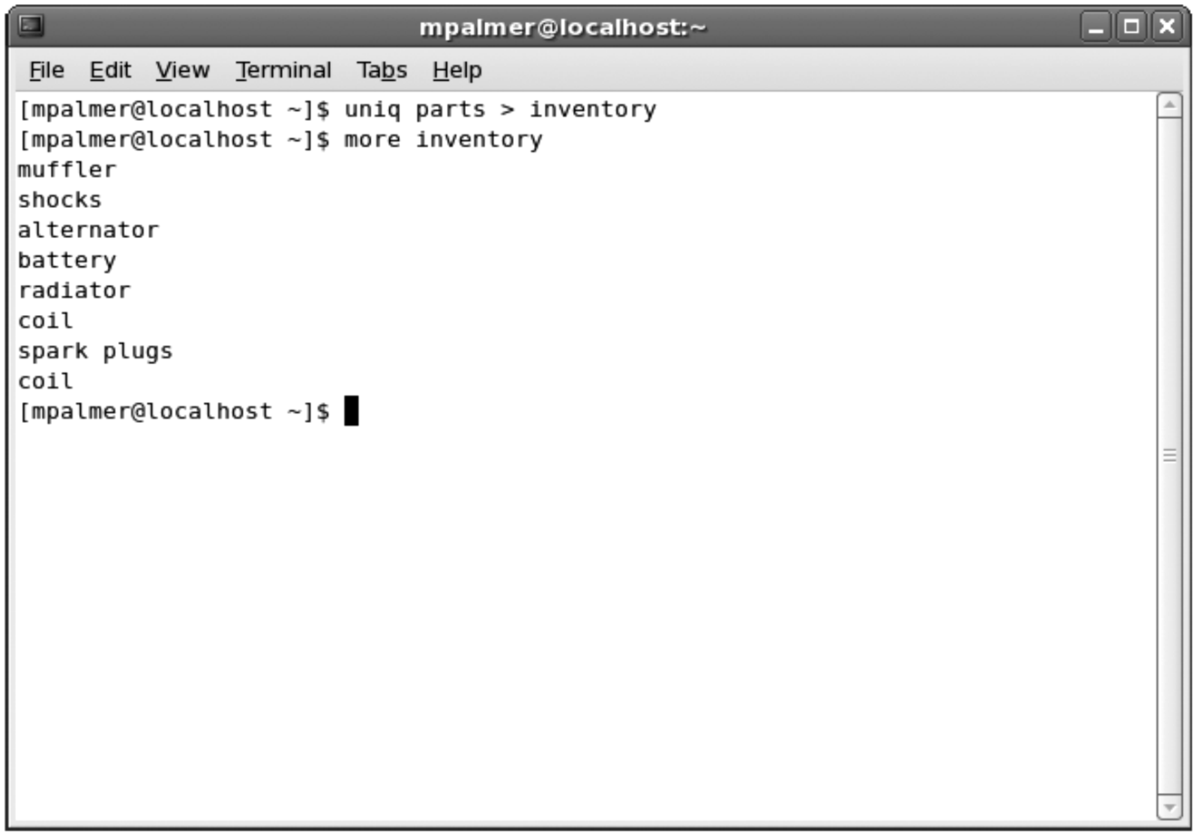
  -*c* starts each line by showing the number of each instance

Using the uniq Command (continued)

- Consider a simple file called parts that contains the following entries:

```
muffler
muffler
shocks
alternator
battery
battery
radiator
radiator
coil
spark plugs
spark plugs
coil
```

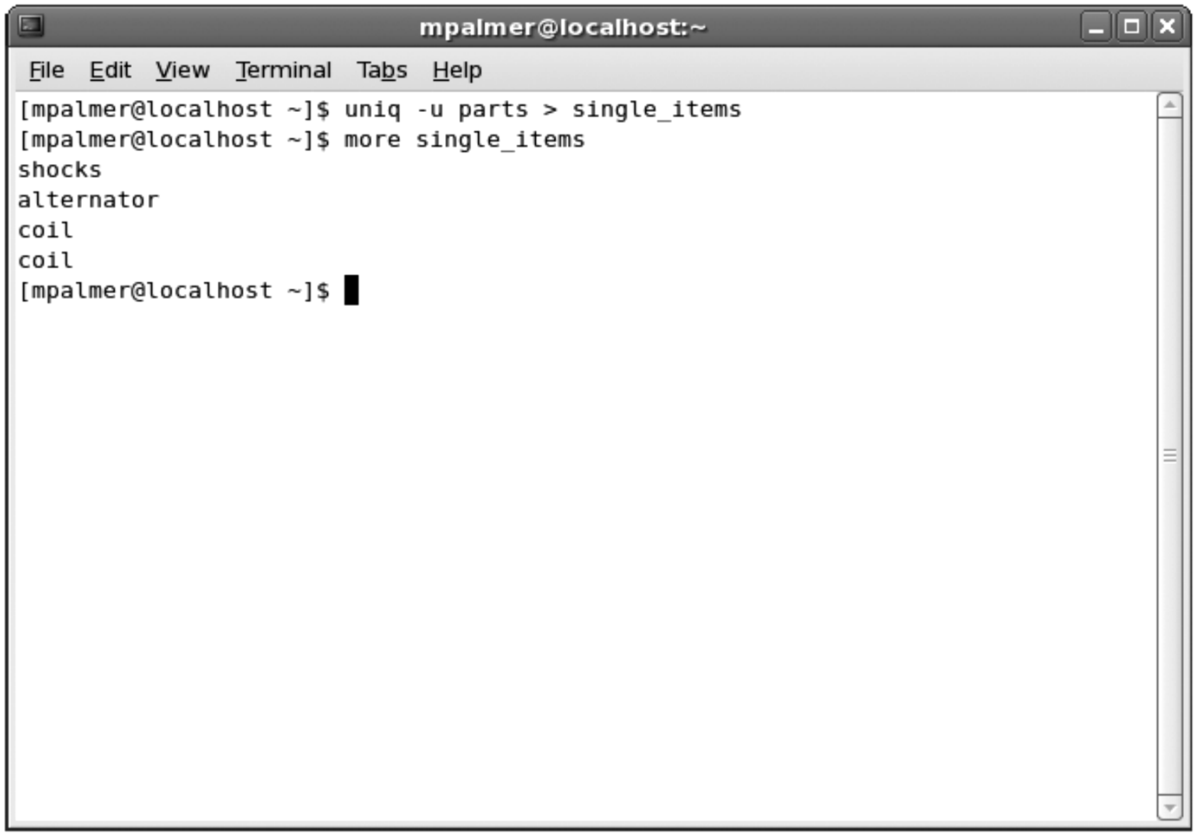
Using the uniq Command (continued)



```
[mpalmer@localhost ~]$ uniq parts > inventory
[mpalmer@localhost ~]$ more inventory
muffler
shocks
alternator
battery
radiator
coil
spark plugs
coil
[mpalmer@localhost ~]$
```

Figure 5-2 Using *uniq* to remove duplicate entries and create a new output file

Using the uniq Command (continued)

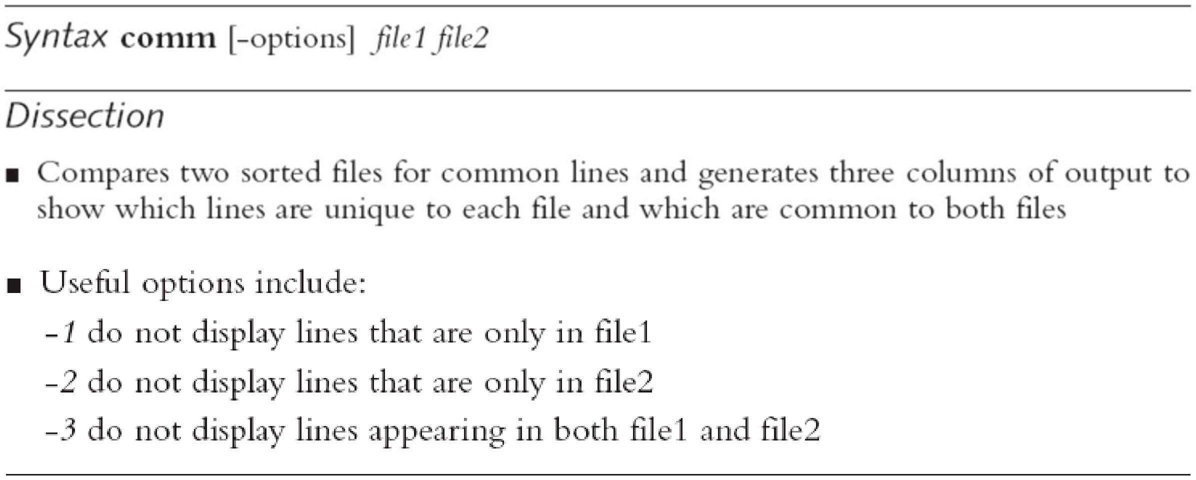


```
[mpalmer@localhost ~]$ uniq -u parts > single_items
[mpalmer@localhost ~]$ more single_items
shocks
alternator
coil
coil
[mpalmer@localhost ~]$
```

Figure 5-3 Creating a file containing only lines not duplicated

Using the comm Command

- Like *uniq*, *comm* identifies duplicate lines
- Unlike *uniq*:
  - Does not delete duplicates
  - Works with two files rather than one



Syntax **comm** [-options] *file1 file2*

Dissection

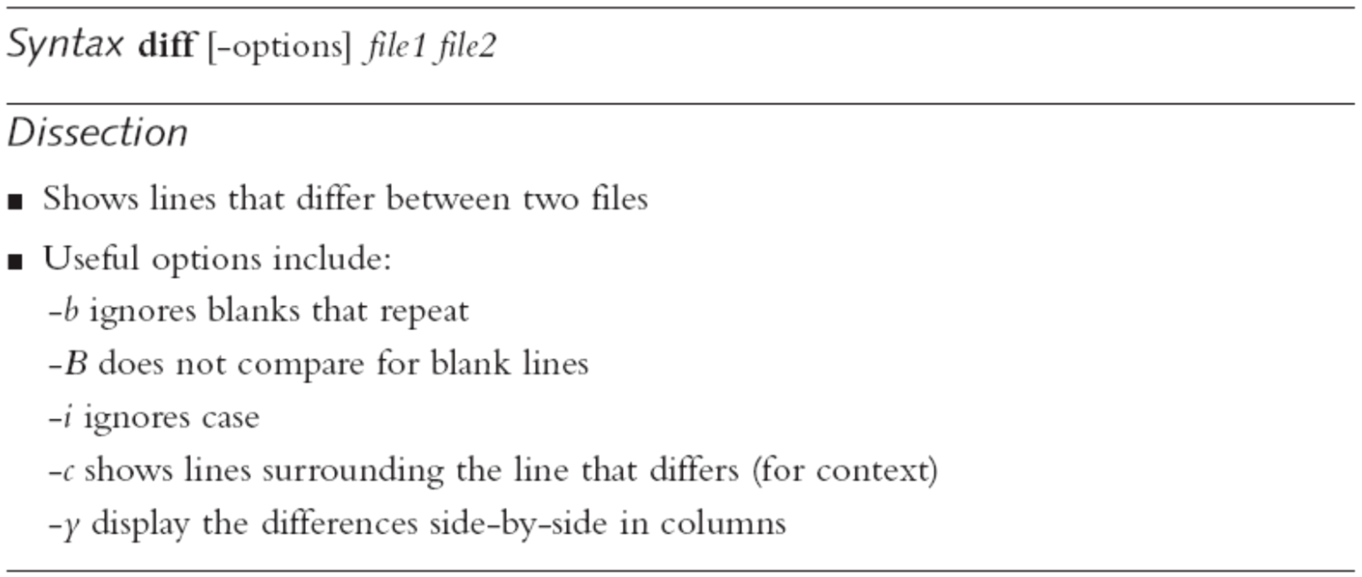
- Compares two sorted files for common lines and generates three columns of output to show which lines are unique to each file and which are common to both files
- Useful options include:

  -*1* do not display lines that are only in file1

  -*2* do not display lines that are only in file2

  -*3* do not display lines appearing in both file1 and file2

Using the diff Command



Syntax **diff** [-options] *file1 file2*

Dissection

- Shows lines that differ between two files
- Useful options include:

  -*b* ignores blanks that repeat

  -*B* does not compare for blank lines

  -*i* ignores case

  -*c* shows lines surrounding the line that differs (for context)

  -*y* display the differences side-by-side in columns

- Commonly used to determine the minimal set of changes needed to convert file1 to file2
- Differing text preceded by < or >

Using the diff Command (continued)

- File zoo1 contains:

```
Monkeys:Bananas:2000:850.00
Lions:Raw Meat:4000:1245.50
Lions:Raw Meat:4000:1245.50
Camels:Vegetables:2300:564.75
Elephants:Hay:120000:1105.75
Elephants:Hay:120000:1105.75
```

- File zoo2 contains:

```
Monkeys:Bananas:2000:850.00
Lions:Raw Meat:4000:1245.50
Camels:Vegetables:2300:564.75
Elephants:Hay:120000:1105.75
```

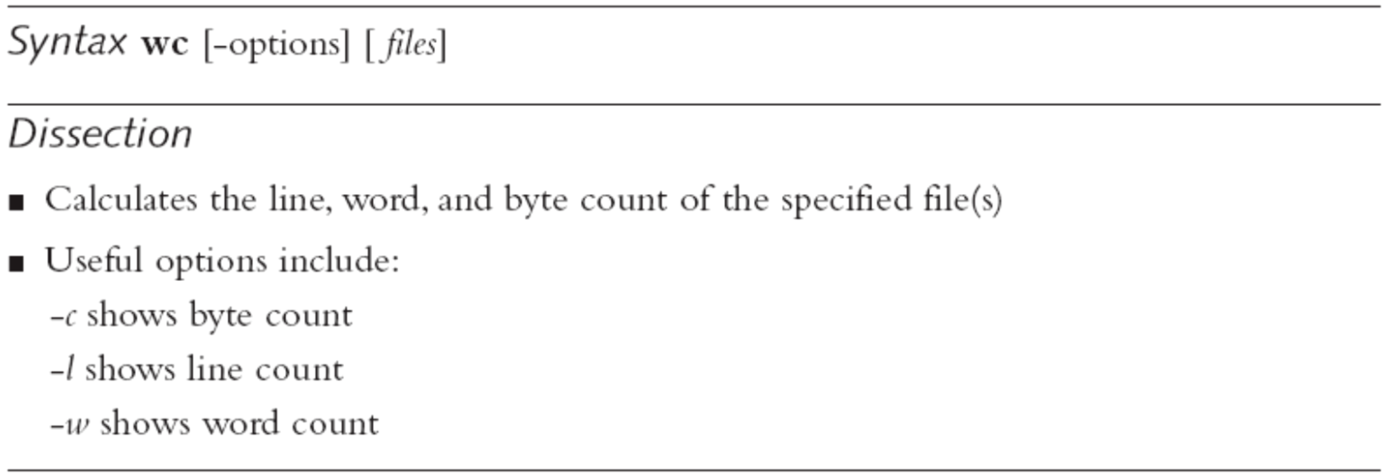
- *diff zoo1 zoo2*

```
3d2
< Lions:Raw Meat:4000:1245.50
…
```

- *diff zoo2 zoo1*

```
2a3
> Lions:Raw Meat:4000:1245.50
…
```

Using the wc Command



Syntax **wc** [-options] [*files*]

Dissection

- Calculates the line, word, and byte count of the specified file(s)
- Useful options include:

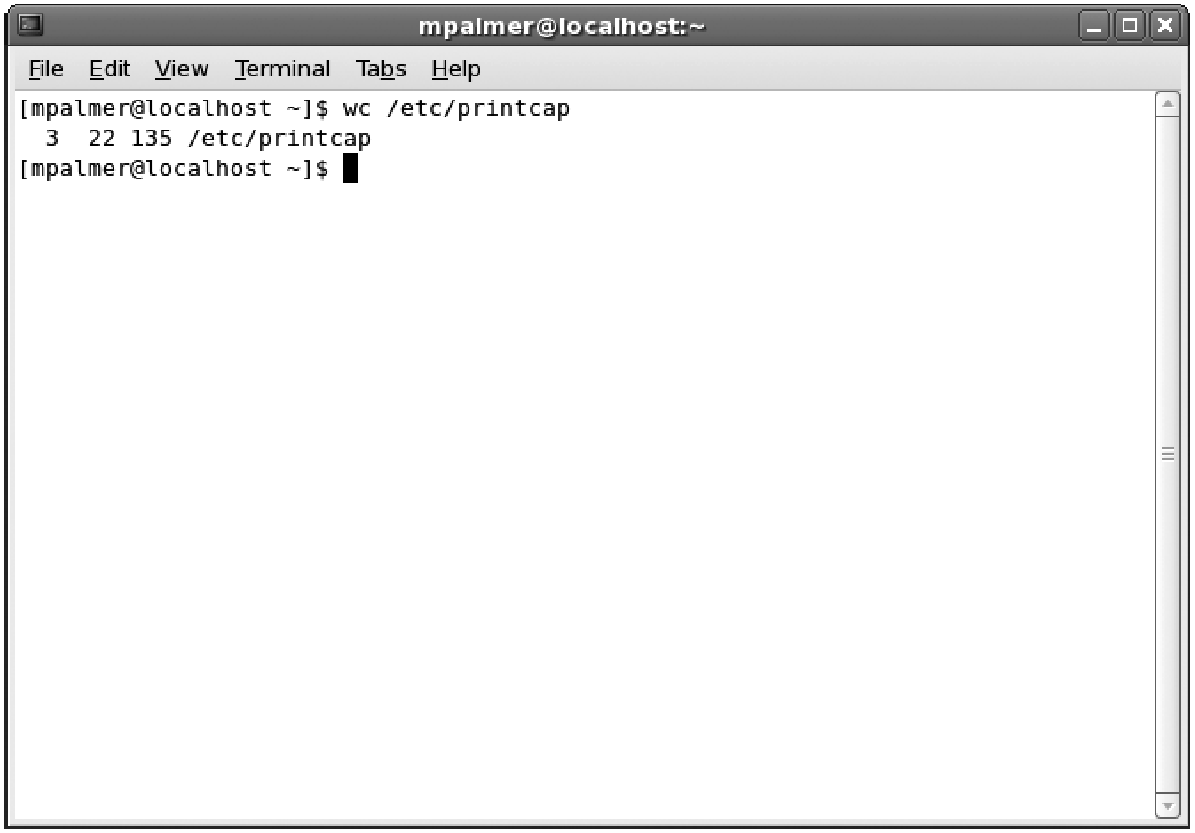
  -*c* shows byte count

  -*l* shows line count

  -*w* shows word count

- You can specify all three options in the command line (e.g., *-lwc*)
- If entered without options, you see counts of lines, words, and characters in that order

Using the wc Command (continued)



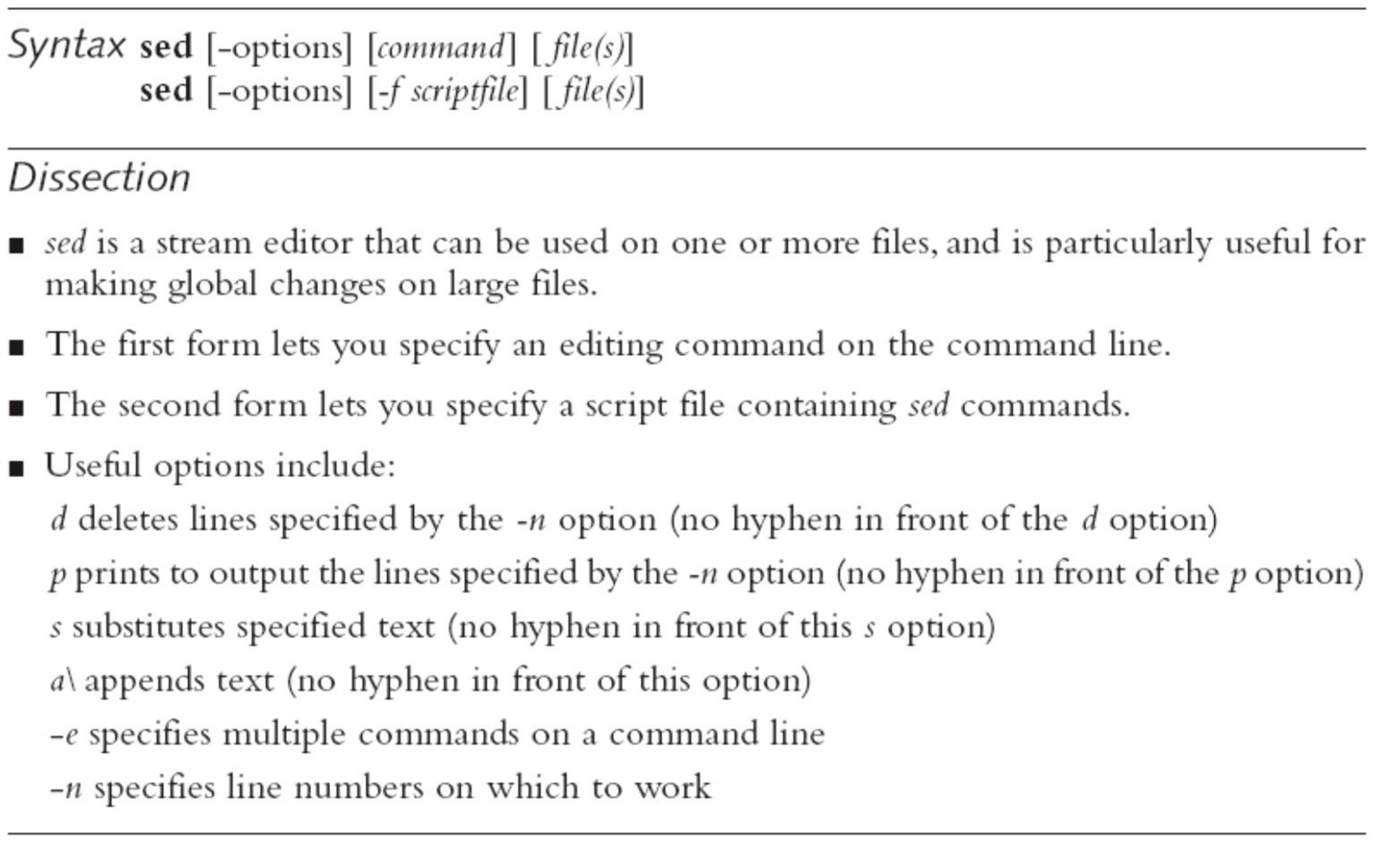
```
[mpalmer@localhost ~]$ wc /etc/printcap
  3  22 135 /etc/printcap
[mpalmer@localhost ~]$ 
```

Figure 5-4 Using *wc* to count lines, words, and bytes in a file

Using Manipulation and Transformation Commands

- Several commands can be used to edit and transform data’s appearance:
  - *sed*
  - *tr*
  - *pr*

Introducing the sed Command



Syntax **sed** [-options] [*command*] [*file(s)*]
sed [-options] [-*f scriptfile*] [*file(s)*]

Dissection

- *sed* is a stream editor that can be used on one or more files, and is particularly useful for making global changes on large files.
- The first form lets you specify an editing command on the command line.
- The second form lets you specify a script file containing *sed* commands.
- Useful options include:

d deletes lines specified by the -*n* option (no hyphen in front of the *d* option)

p prints to output the lines specified by the -*n* option (no hyphen in front of the *p* option)

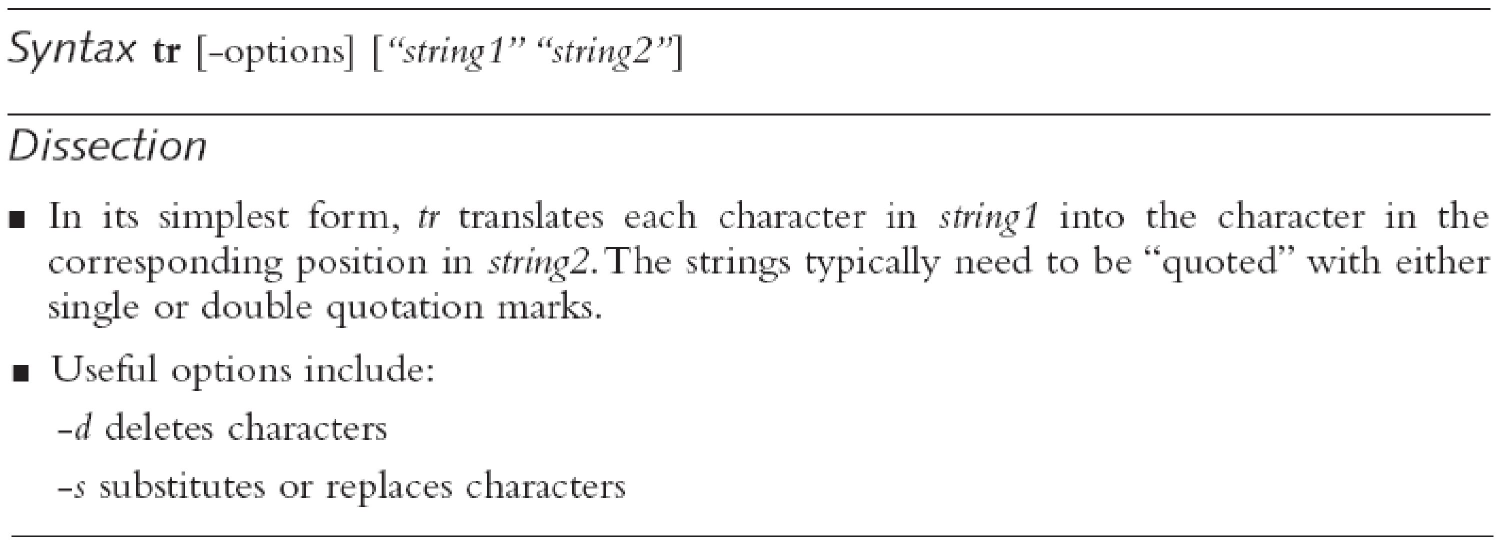
s substitutes specified text (no hyphen in front of this *s* option)

a\ appends text (no hyphen in front of this option)

–*e* specifies multiple commands on a command line

–*n* specifies line numbers on which to work

Translating Characters Using the tr Command



Syntax **tr** [-options] [*“string1” “string2”*]

Dissection

- In its simplest form, *tr* translates each character in *string1* into the character in the corresponding position in *string2*. The strings typically need to be “quoted” with either single or double quotation marks.
- Useful options include:

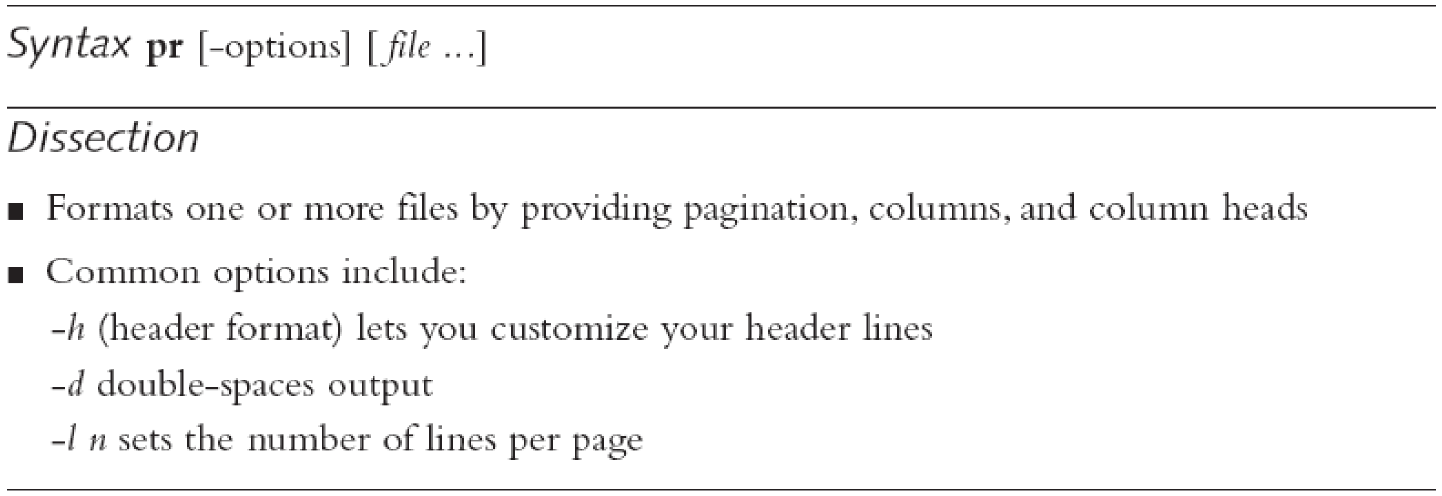
  -*d* deletes characters

  -*s* substitutes or replaces characters

- A popular use is to convert lowercase characters to uppercase characters
- Some examples:

```
tr "c" " " < constants
tr ’c’ ’ ’ < constants
```

Using the pr Command to Format Your Output



Syntax **pr** [-options] [*file* ...]

Dissection

- Formats one or more files by providing pagination, columns, and column heads
- Common options include:

  -*h* (header format) lets you customize your header lines

  -*d* double-spaces output

  -*l n* sets the number of lines per page

- If no file is specified or "-" is specified as the file, reads the standard input
- Default output: single-column pages of 66 lines
  – Each page has a five-line header

Designing a New File Processing Application

- Files, records, and fields are **logical structures**
- How you set up records in a file can influence what you can do with an application
  - Also affects the ways in which you can use selection/manipulation/transformation commands
- Selection of fields is important for enabling useful sorts and for linking of files (*join*)
  - An ID can be a key field for sorting

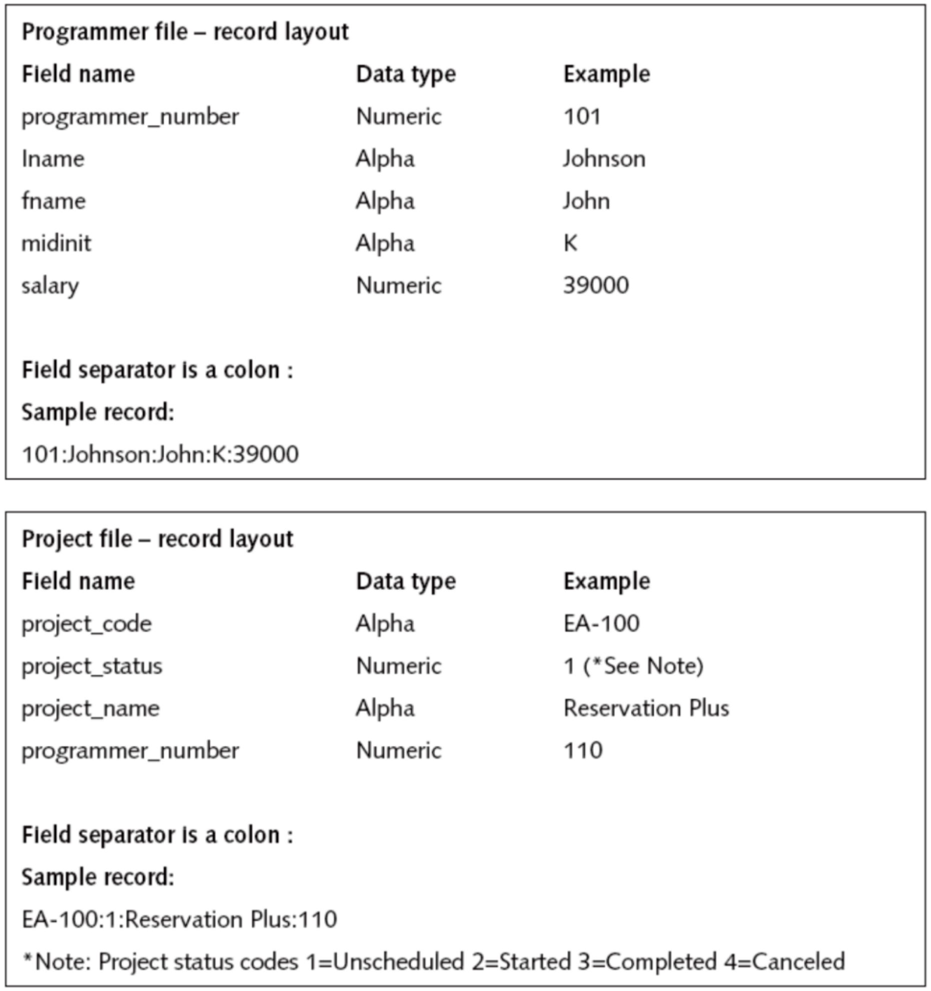
Designing Records

- First task in record design phase: define fields
  - **Record layout** identifies each field by name and data type
- Design file record to store only those fields relevant to each record’s primary purpose
- Short records are preferable
- Must include a field that uniquely identifies each record in the file

Linking Files with Keys

- Multiple files can be joined by a key
  - Key: common field shared by each of the linked files
- Plan a way to join files in design phase

Linking Files with Keys (continued)



Programmer file – record layout

Field name	Data type	Example
programmer_number	Numeric	101
lname	Alpha	Johnson
fname	Alpha	John
midinit	Alpha	K
salary	Numeric	39000

Field separator is a colon :

Sample record:

101:Johnson:John:K:39000

Project file – record layout

Field name	Data type	Example
project_code	Alpha	EA-100
project_status	Numeric	1 (*See Note)
project_name	Alpha	Reservation Plus
programmer_number	Numeric	110

Field separator is a colon :

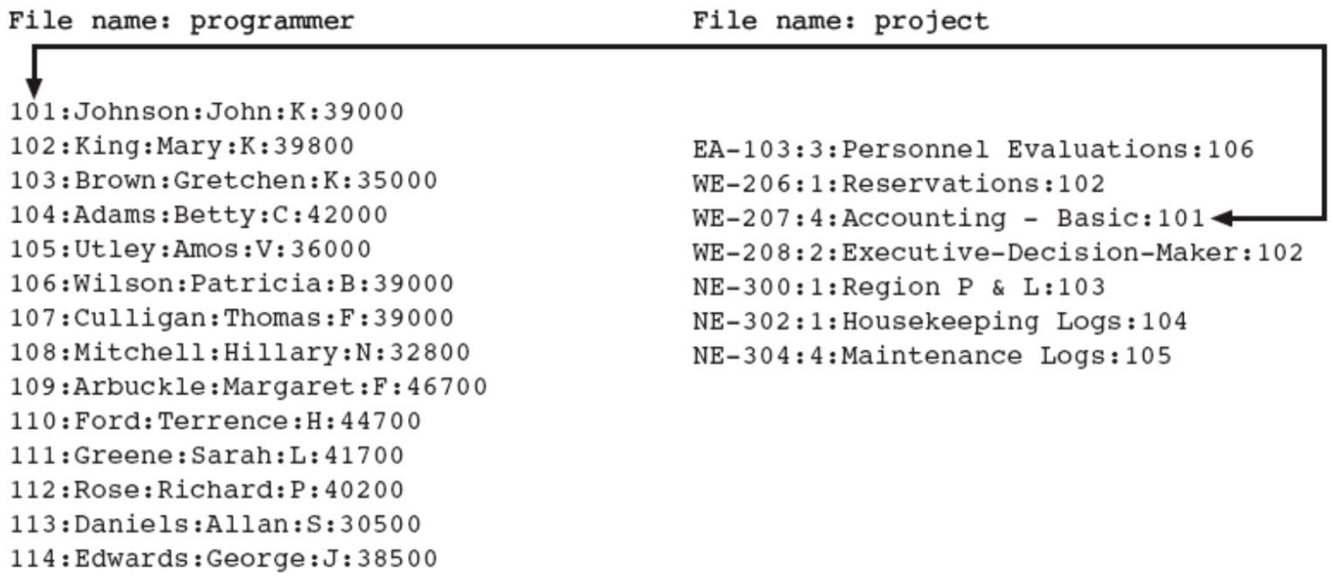
Sample record:

EA-100:1:Reservation Plus:110

*Note: Project status codes 1=Unscheduled 2=Started 3=Completed 4=Canceled

Figure 5-5 Programmer and project file record layouts

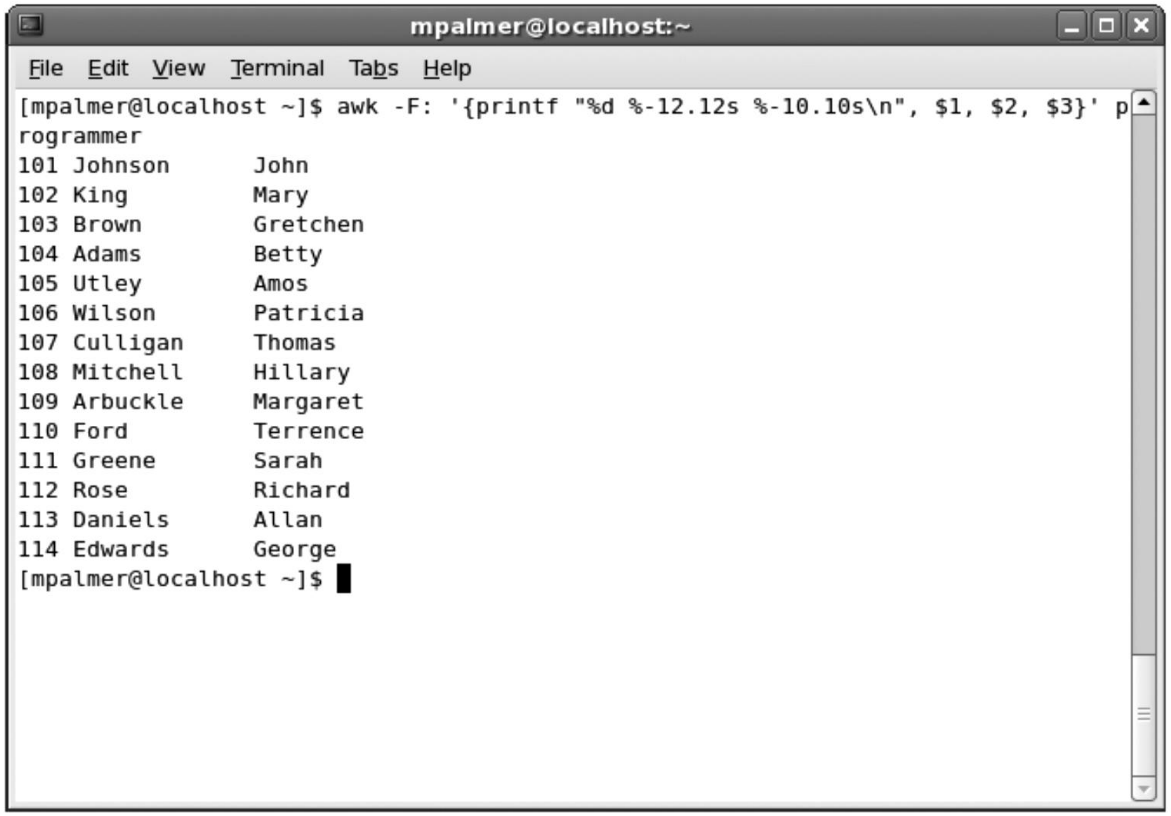
Creating the Programmer and Project Files



```
File name: programmer                      File name: project

101:Johnson:John:K:39000
102:King:Mary:K:39800                      EA-103:3:Personnel Evaluations:106
103:Brown:Gretchen:K:35000                 WE-206:1:Reservations:102
104:Adams:Betty:C:42000                    WE-207:4:Accounting - Basic:101
105:Utley:Amos:V:36000                     WE-208:2:Executive-Decision-Maker:102
106:Wilson:Patricia:B:39000                NE-300:1:Region P & L:103
107:Culligan:Thomas:F:39000                NE-302:1:Housekeeping Logs:104
108:Mitchell:Hillary:N:32800               NE-304:4:Maintenance Logs:105
109:Arbuckle:Margaret:F:46700
110:Ford:Terrence:H:44700
111:Greene:Sarah:L:41700
112:Rose:Richard:P:40200
113:Daniels:Allan:S:30500
114:Edwards:George:J:38500
```

Formatting Output (continued)



```
[mpalmer@localhost ~]$ awk -F: '{printf "%d %-12.12s %-10.10s\n", $1, $2, $3}' p
rogrammer
101 Johnson      John
102 King         Mary
103 Brown        Gretchen
104 Adams        Betty
105 Utley        Amos
106 Wilson       Patricia
107 Culligan     Thomas
108 Mitchell     Hillary
109 Arbuckle     Margaret
110 Ford         Terrence
111 Greene       Sarah
112 Rose         Richard
113 Daniels      Allan
114 Edwards      George
[mpalmer@localhost ~]$ 
```

Figure 5-7 *awk* report using *printf* to display the three fields

Using a Shell Script to Implement the Application

- Tip: test and debug each command before you place it in your script file
- Use vi or Emacs to create script files
- Commenting shell scripts is crucial
  - Helps creator and other programmers
  - Use pound (#) character

Running a Shell Script

- You can run a shell script in virtually any shell
  - We will use Bash
- Two easy ways to run scripts:
  - Call the interpreter: `sh testscript`
    - Can accompany it with several debugging options
  - Type ./ in front of name: `./testscript`
    - Must make script executable first
      - Use *chmod* to add x permission
- Advice: specify with what shell your script is intended to be used
  - Example: *#!/bin/bash*

Putting It All Together to Produce the Report

- Combine small scripts into a larger script file
  - Convenient
  - Complete a large task by dividing it into a series of smaller ones
  - Test each small script independently

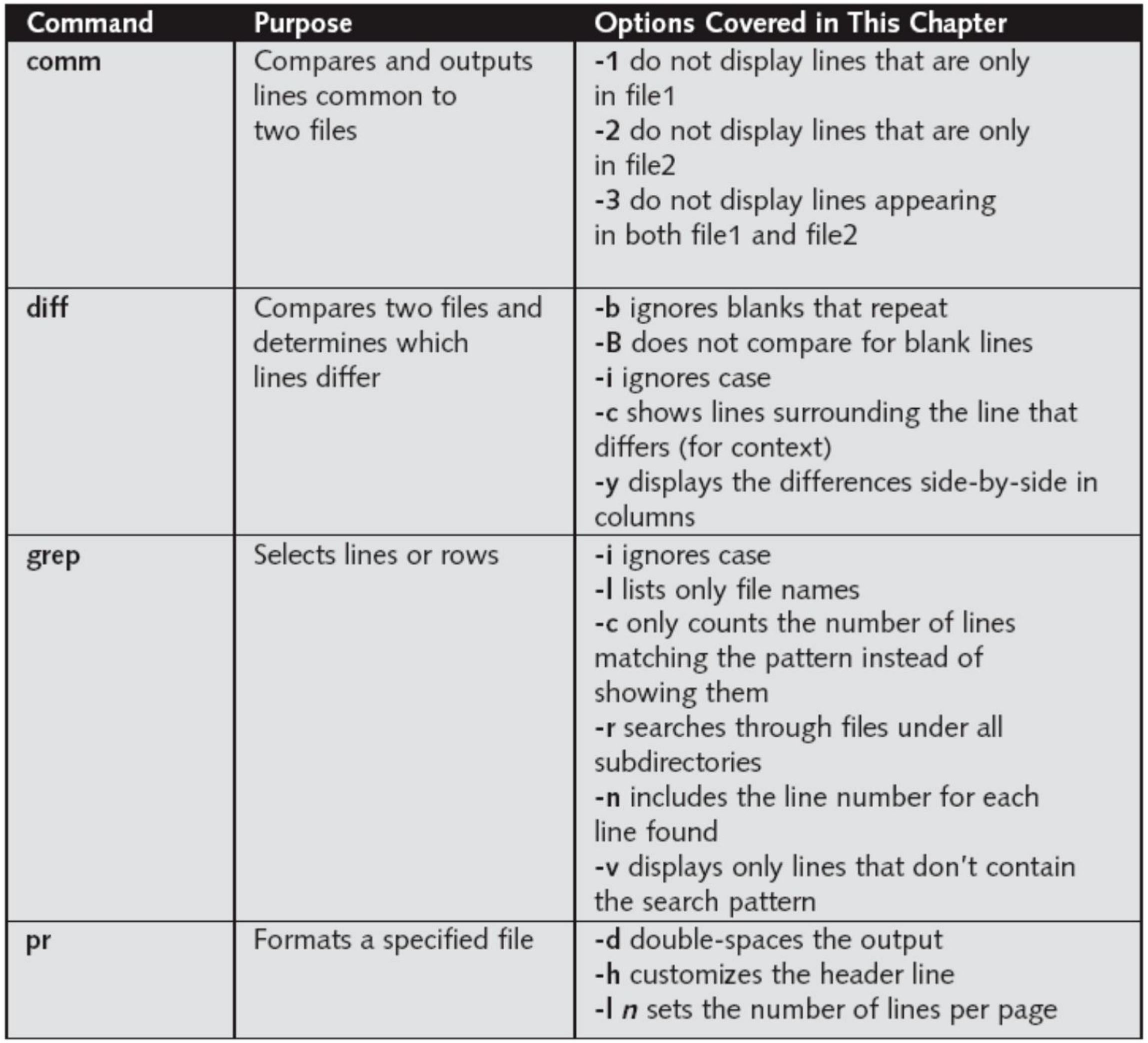
Summary

- Selection commands extract information
- Manipulation and transformation commands alter extracted information into useful/appealing formats
- *grep* searches for a specific pattern in a file
- *uniq* removes duplicate lines from a file
- *comm* compares lines common to two different files and produces three-column output with the variances
- *diff* attempts to determine the minimum set of changes needed to convert the contents of one file to match the contents of another file

Summary (continued)

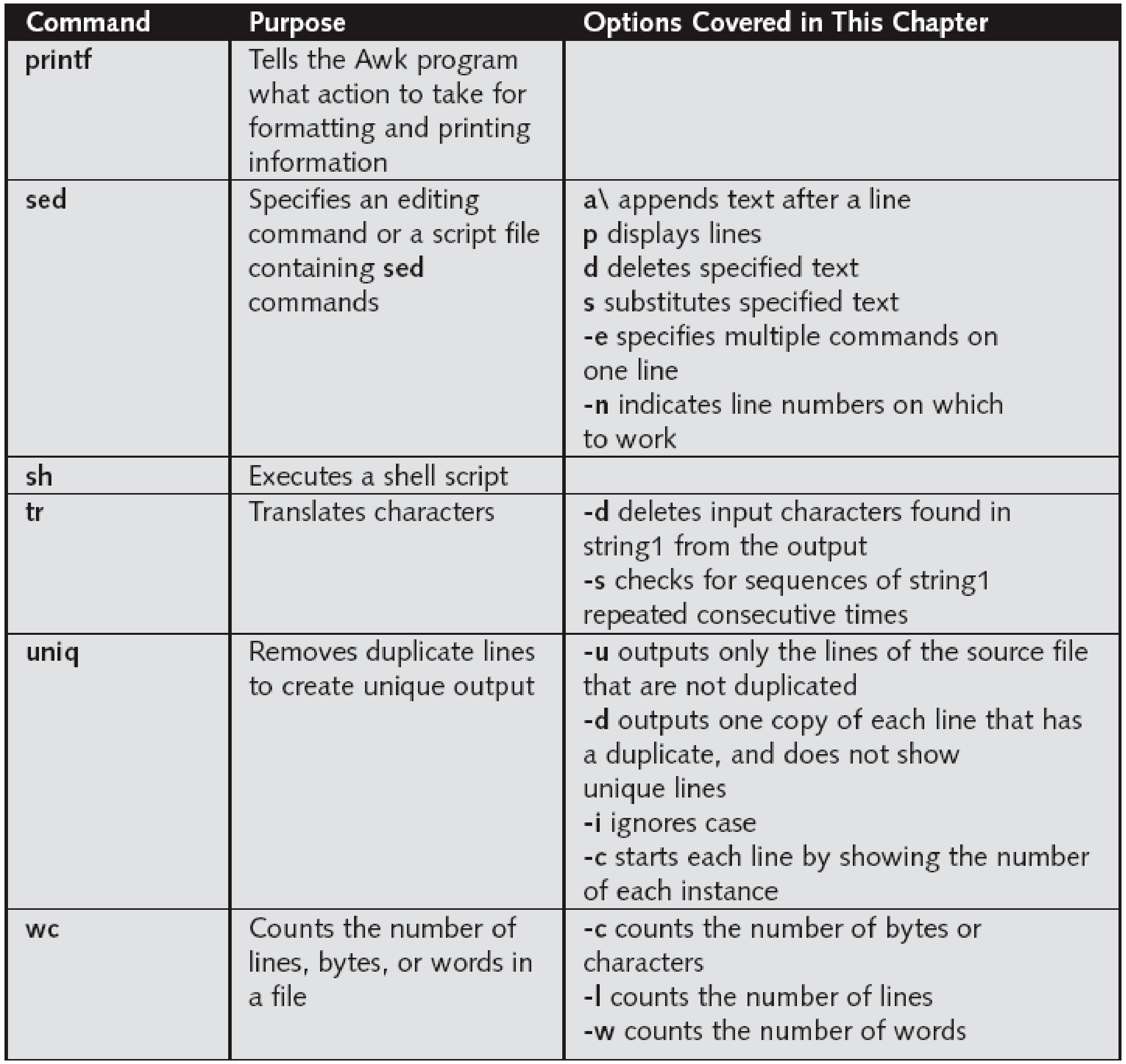
- *wc* counts bytes, words, or lines in a file
- *sed* is a stream editor designed to make global changes to large files
- *tr* copies data read from the standard input to the standard output, substituting or deleting the characters specified by options and patterns
- *pr* prints the standard output in pages
- When designing a file-processing application, define logical structures (including record layout)
- Shell scripts should be commented and simple

Command Summary



Command	Purpose	Options Covered in This Chapter
comm	Compares and outputs lines common to two files	**-1** do not display lines that are only in file1 **-2** do not display lines that are only in file2 **-3** do not display lines appearing in both file1 and file2
diff	Compares two files and determines which lines differ	**-b** ignores blanks that repeat **-B** does not compare for blank lines **-i** ignores case **-c** shows lines surrounding the line that differs (for context) **-y** displays the differences side-by-side in columns
grep	Selects lines or rows	**-i** ignores case **-l** lists only file names **-c** only counts the number of lines matching the pattern instead of showing them **-r** searches through files under all subdirectories **-n** includes the line number for each line found **-v** displays only lines that don't contain the search pattern
pr	Formats a specified file	**-d** double-spaces the output **-h** customizes the header line **-l** ***n*** sets the number of lines per page

Command Summary (continued)



Command	Purpose	Options Covered in This Chapter
printf	Tells the Awk program what action to take for formatting and printing information	
sed	Specifies an editing command or a script file containing **sed** commands	**a** appends text after a line **p** displays lines **d** deletes specified text **s** substitutes specified text **-e** specifies multiple commands on one line **-n** indicates line numbers on which to work
sh	Executes a shell script	
tr	Translates characters	**-d** deletes input characters found in string1 from the output **-s** checks for sequences of string1 repeated consecutive times
uniq	Removes duplicate lines to create unique output	**-u** outputs only the lines of the source file that are not duplicated **-d** outputs one copy of each line that has a duplicate, and does not show unique lines **-i** ignores case **-c** starts each line by showing the number of each instance
wc	Counts the number of lines, bytes, or words in a file	**-c** counts the number of bytes or characters **-l** counts the number of lines **-w** counts the number of words