Expr Examples

```
var=`expr “$var” + 1`
                # add 1 to var

if  [ `expr “$s1” \< “$s2”` = 1 ]
                # check if $s1 < $s2, works
                # for strings as well

a=`expr “$a” \* 2`
                # double the value of a
```

Awk (20.10)

- Awk is a Unix utility that can manipulate text files that are arranged in columns. Like most other Unix utilities, it reads from standard input if no files are specified.
- General forms:

  awk [-F*c*] 'script' [files]

  awk [-F*c*] -f scriptfile [files]
- -F*c* indicates a field separator to be the specified character *c*. For instance, one could specify -F: to indicate that ':' is the field separator. By default the field separator is blanks and tabs.

Awk Scripts (20.10)

- Each script consist of one or more pairs of:

  pattern { procedure }
- Awk processes each line of a file, one at a time. For each pattern that matches, then the corresponding procedure is performed.
- If the pattern is missing, then the procedure is applied to each line.
- If the { procedure } is missing, then the matched lines are written to standard output.

Awk Patterns (20.10)

- Awk patterns can be any of the following. Awk patterns, except for BEGIN and END, can be combined using the logical operators ||, &&, and !.
  - / regular expression /
  - relational expression
  - pattern-matching expression
  - BEGIN
  - END

Awk Regular Expressions

- A pattern that uses a regular expression indicates that somewhere in the line the regular expression matches. The regular expression returns true or false depending on if it matches within the current line. Using a pattern only (causes each line to be printed that matches) results in similar functionality as grep.

```
/D[Rr]\./ # matches any line containing DR. or Dr.
/^#/      # matches any line beginning with '#'
```

Awk Relational Expressions (20.10)

- An awk relational expression can consist of strings, numbers, arithmetic/string operators, relational operators, defined variables, and predefined variables.
  - $0 means the entire line
  - $n means the nth field in the current line
  - NF is a predefined variable indicating the number of fields in the current line
  - NR is the number of the current line

Example Awk Relational Exprs (20.10)

```
$1 == "if"            # Is first field an "if"?
$1 == $2              # Are first two fields the same?
NR > 100              # Have already processed 100 lines?
NF > 5                # Does current line have > 5 fields?
NF > 5 && $1 == $2    # Compound condition.
/if/ && /</           # Does line contain "if" and a "<"?
/while {/ || /do {/   # Does line contain "while {" or "do {"
```

Awk Pattern-Matching Exprs (20.10)

- Pattern matching expressions can check if a regular expression matches (~) or does not match (!~) a field. Note that the pattern does not have to match the entire field.

```
$1 ~ /D[Rr]\./ # First field match "DR." or "Dr."?
$1 !~ /#/      # First field does not match "#"?
```

Awk BEGIN and END Patterns (20.10)

- The BEGIN pattern lets you specify a procedure before the first input line is processed.
- The END pattern lets you specify a procedure after the last input line is processed.

Awk Procedures (20.10)

- An awk procedure specifies the processing of a line that matches a given pattern. An awk procedure is contained within the '{' '}' and consists of commands separated by semicolons or newlines.
- Commonly used awk commands include:

```
print [args]    # print arguments, if none print $0
var = expr      # assignment to variables
```

Example Awk Scripts (20.10)

```
{ print $1, $2 }          # print the first two fields of each line

$0 !~ /^$/                # print lines with one or more chars

$2 > 0 && $2 < 10         # print second field when its value
{print $2}                # is in the range 1..9

BEGIN {sum=0}             # sums the values of the fields in the
{sum += $1}               # first column and prints the sum
END {print sum}
```

Tr (21.11)

- The *tr* unix utility is a character *tr*anslation filter. The most common usage is to replace one character for another. It reads from standard input and writes to standard output.
- General forms.

```
tr string1 string2        # replace character in string1 with
                          # corresponding character in string2

tr -d string              # delete all occurrences of characters
                          # that appear in string

tr -s string1 string2
                          # replace instances of repeated chars
                          # in string1 with a single char in
                          # string2
```

Tr Examples

```
tr “[a-z]” “[A-Z]”      # convert lower case to uppercase
tr '&' '#'              # convert &'s to #'s
tr -s “\t” “\t”         # squeeze consecutive tabs to a
                        # single tab
tr -d '\015'            # deletes the carriage return
                        # characters from a DOS file
```

Basename (36.13)

- The basename utility strips off the portion of a pathname that precedes and including the last slash and prints the result to standard output.

```
% basename /home/faculty/whalley/cop4342exec/plot.p
plot.p
```

```
# The gcc script needs to strip off the path and create a file called “echo.o”
# in the current directory.
% gcc -c /home/faculty/whalley/test/src/echo.c
```

Dirname (36.13)

- The dirname utility strips off the last slash and everything that follows it in the pathname.

```
% dirname /home/faculty/whalley/cop4342exec/plot.p
/home/faculty/whalley/cop4342exec
```

- Often shell scripts will determine the directory in which they reside.

```
dir=`dirname $0`
if $dir/database_exists.sh $1
then
  ...
fi
```

Sort (22.2)

- Sort divides each line into fields separated by a whitespace (blanks or tabs) character and sorts the lines by field, from left to right. It writes to standard output.

- General form. The -b option causes leading blank characters for each field to be ignored. The -r option causes the sort to be in reverse order. The +pos# indicates the position of the next field to start sorting. The -pos# indicates the position of the field to stop sorting. Field positions are numbered starting from 0. If no files are specified, then it sorts the standard input. If more than one file is specified, then it sorts all the files together.

  sort [-b] [-r] [+pos# [-pos#]]* [files]

Sort Examples

- Say the file “data.txt” contains the following:

```
Smith Bob 27 32312
Jones Carol 34 32306
Miller Ted 27 32313
Smith Alice 34 32312
Miller Ted 45 32300
```

- “sort data.txt” would produce:

```
Jones Carol 34 32306
Miller Ted 27 32313
Miller Ted 45 32300
Smith Alice 34 32312
Smith Bob 27 32312
```

Sort Examples (cont.)

- “sort +2 data.txt” would produce:

```
Smith Bob 27 32312
Miller Ted 27 32313
Jones Carol 34 32306
Smith Alice 34 32312
Miller Ted 45 32300
```

- “sort +2 -3 +0 data.txt” would produce:

```
Miller Ted 27 32313
Smith Bob 27 32312
Jones Carol 34 32306
Smith Alice 34 32312
Miller Ted 45 32300
```

Sort Examples (cont.)

- “sort +3 data.txt” would produce:

```
Miller Ted 45 32300
Jones Carol 34 32306
Smith Alice 34 32312
Smith Bob 27 32312
Miller Ted 27 32313
```

- “sort +1 -2 +0 -1 +3 data.txt” would produce:

```
Smith Alice 34 32312
Smith Bob 27 32312
Jones Carol 34 32306
Miller Ted 45 32300
Miller Ted 27 32313
```

Reverse Sort (22.6)

- Use the -r option to reverse the order of the sort.
- “sort -r < data.txt” would produce:

```
Smith Bob 27 32312
Smith Alice 34 32312
Miller Ted 45 32300
Miller Ted 27 32313
Jones Carol 34 32306
```

Type of Sort (22.5)

- By default, sort orders the lines in alphabetical (actually ASCII) order. You can specify that a particular field is to be sorted in numerical order by including an 'n' after the field position.
- Say the file “data2.txt” contained:

```
Smith 42
Jones 7
Miller 100
```

- “sort +1” produces and “sort +1n” produces

```
Miller 100          Jones 7
Smith 42            Smith 42
Jones 7             Miller 100
```

Ignoring Extra Whitespace (22.6)

- The -b option indicates to ignore extra whitespace before each field. But you have to specify the fields that you want to sort.
- Say we have the following file called “data.txt”.

```
 Smith Bob 27 32312
Jones Carol 34 32306
Miller Ted 27 32312
Smith Alice 34 32312
Miller Ted  45 32300
```

- What would “sort < data.txt” produce?
- How could you get an ordering where extra whitespace is ignored and the age field is sorted numerically?

Latex

- Latex is document markup language and document preparation system.
- It is used as a text formatter (batch) as opposed to a WYSIWYG editor (interactive), like MS Word.
- Latex converts a text file with embedded latex commands into a dvi file.
- Below is the general form of a Latex file.

```
\documentclass{class}
\begin{document}
...
\end{document}
```

- Example invocation.

```
latex  tmp.tex              # produces a tmp.dvi file
dvipdf -o tmp.dvi tmp.dvi   # produces a tmp.ps file
```

Latex Table Specification Format

- Multiple tables may appear in a latex document.
- Table specification format:

```
\begin{table}
table alignment
\begin{tabular}{column format}
data to be laid out in the table
...
data to be laid out in the table
\end{tabular}
\end{table}
```

Latex Table Example

- Assume the following text in a *tmp.tex* file.

```
\documentclass{article}              % document class article
\pagestype{empty}                    % no page numbers
\begin{document}                     % start of document
\begin{table}                        % start of table
\centering                           % center table
\begin{tabular}{|l|l|r|c|}           % specify column format
\hline                               % line at top of table
Last & First & Age & Zipcode\\       % headings
\hline                               % line under headings
Jones & Carol & 34 & 32306\\
Miller & Ted & 27 & 32313\\
Miller & Ted & 45 & 32300\\
Smith & Alice & 34 & 32312\\
Smith & Bob & 27 & 32312\\
\hline                               % line at end of table
\end{tabular}                        % end of tabular
\end{table}                          % end of table
\end{document}                       % end of document
```

Latex Example (cont.)

- One can view a postscript file with ghostview. Issuing the following commands will result in the table being displayed.

```
latex tmp.tex
dvipdf tmp.dvi tmp.pdf
acroread tmp.pdf
```

Last	First	Age	Zipcode
Jones	Carol	34	32306
Miller	Ted	27	32313
Miller	Ted	45	32300
Smith	Alice	34	32312
Smith	Bob	27	32312

Another Latex Table Example

- Assume the following text in a *tmp2.tex* file.

```
\documentclass{article}                 % document class article
\pagestyle{empty}                       % no page numbers
\usepackage{multirow}                   % used for spanning rows
\begin{document}                        % start of document
\begin{table}                           % start of table
\centering                              % center table
\begin{tabular}{|l|l|r|c|}              % specify column format
\hline                                  % line at top of table
\multicolumn{2}{|c|}{Name} & \multirow{2}{*}{Age} &
\multirow{2}{*}{Zipcode}\\              % headings
\cline{1-2}                             % line under first two cols
Last & First & & \\
\hline                                  % line under headings
Jones & Carol & 34 & 32306\\
...
Smith & Bob & 27 & 32312\\
\hline                                  % line at end of table
\end{tabular}                           % end of tabular
\end{table}                             % end of table
\end{document}                          % end of document
```

Another Latex Example (cont.)

- Issuing the following commands will result in the previous table being displayed.

```
latex tmp2.tex
dvipdf tmp2.dvi tmp2.pdf
acroread tmp2.pdf
```

Name		Age	Zipcode
Last	First		
Jones	Carol	34	32306
Miller	Ted	27	32313
Miller	Ted	45	32300
Smith	Alice	34	32312
Smith	Bob	27	32312

Fmt Utility (21.2)

- The fmt utility is a simple text formatter that fills and joins lines to produce output up to the number of characters specified or 72 by default.
- General form.

  fmt [-w *width*] [*inputfiles*]

Fmt Utility (21.2)

- The fmt utility is a simple text formatter that fills and joins lines to produce output up to the number of characters specified or 72 by default.
- General form.

  fmt [-w *width*] [*inputfiles*]

Fmt Example

- Say you have the following text in a *tmp.txt* file.

```
Four score and seven years ago our fathers
brought forth on this continent, a new nation,
conceived in Liberty, and dedicated to the
proposition that all men are created equal.
```

- Using the command “fmt -w 30 tmp.txt”, the output would be:

```
Four score and seven years
ago our fathers brought
forth on this continent,
a new nation, conceived in
Liberty, and dedicated to the
proposition that all men are
created equal.
```

Cut (21.14)

- Cut allows you to select a list of fields from one or more files. If no files are specified, then it reads from standard input. It writes to standard output.
- General form. The -d option allows you to change the field delimiter. By default it is a tab. The -f indicates a list of fields to be cut. Fields are numbered starting from 1. The list can also include ranges of fields. At least one field must be specified.

  cut [-d“<*char*>”] -f<*num*>[,<*num*>]* [*file*]*

Cut Examples

cut -d“ ” -f1 tmp.db	# print the first field from tmp.db
cut -f2,4 data.txt	# print the 2nd and 4th fields from data.txt
cut -d: -f1 /etc/passwd	# print the user id's from the passwd file
cut -f1,3-5 data.txt	# print 1st, 3rd,4th, 5th fields from data.txt

Paste (21.18)

- The paste utility merges files together by concatenating the corresponding lines of the specified input files. It writes to standard output. If one input file is not as long as another, then paste will treat the shorter file as having empty lines at the end. By default, paste replaces each newline character with a tab, except for the newline character in the last file. You can replace the tab with some other character by using the -d option.
- General form.

  paste [-d<char>] files

Paste Example

- Paste is often used together with the cut utility.

```
# cut the first field of file1.txt and file2.txt
cut -f1 file1.txt > tmp1.txt
cut -f1 file2.txt > tmp2.txt

# paste these fields together separated by a blank
paste -d“ ” tmp1.txt tmp2.txt > file3.txt

# remove temporary files
rm tmp1.txt tmp2.txt
```

Head (12.12) and Tail (12.8-12.10)

- Head prints the first few lines of one or more files. Tail prints the last few lines of a file.
- General form. If -<*num*> is not specified, then 10 lines are printed by default. The -f option indicates that tail will monitor the *file* every second and update it if it changes.

  head [-<*num*>] [*files*]

  tail [-<*num*>] [-f] [*file*]