* **Lecture 12 - Variable Expansion; Shell Expansion; Shell Startup; alias; Odds and Ends**

**Variable Expansion**

* + $var - "$" before a variable name will retrieve it's value, also called variable substitution
  + ${var} - brace brackets can be used to delineate the variable name from following characters
  + ${!var} - will use value of "var" as the variable whose value will be retrieved - this is called indirect expansion:

==> set one two three four five

==> num=3

﻿==> echo ${num}3

﻿==> echo ${!num}three

==> varName=num

==> echo ${varName}num

==> echo ${!varName}3

﻿==> \_

* + note that this can also be done with eval :

==> set one two three four five

==> num=3

﻿==> echo ${num}

3

﻿==> echo$${num}

 7395{num}

==> echo \$${num}

$3

﻿==> eval echo \Error converting from LaTeX to MathML{varName}

3

﻿==> \_

* + note that combining eval with indirect expansion gives a double indirection:

==> echo ${varName}

num

==> echo ${!varName}

3

﻿==> eval echo \$${!varName}

three

==> \_

* + indirect expansion can be used to get the value of the last positional parameter:

==> set one two three four five

==> args=$#

﻿==> lastarg=${!args}

==> echo $lastarg

﻿fiv﻿e

==> echo ${!#}

five

﻿==> \_

* + ${var:-default} - expands to "default" value if "var" is null or unset, "var" is unchanged
  + ${var:+default} - expands to "default" value if "var" is set, "var" is unchanged
  + ${var:?errmsg} - sends "errmsg" to stderr if "var" is null or unset, "var" is unchanged, if the shell is non-interactive it will exit
  + ${var:=default} - sets "var" to "default" value if "var" is null or unset
  + here are the results when the variable is not set:

==> echo $var

﻿==> echo ${var:-"this default value will be displayed"}

this default value will be displayed

==> echo $var

﻿==> echo ${var:+"this default value will not be displayed"}

==> echo ${var:?}

-bash: var: parameter null or not set

==> echo ${var:?"this error message will be sent to stderr"}

-bash: var: this error message will be sent to stderr

==> echo $?

1

﻿==> echo ${var:="var now has this value"}

var now has this value

==> echo $var

﻿var now has this value

==> \_

* + without the colon ":", the variable would be checked for existence only, the value would not be checked for null
  + here are the results with the variable set:

==> echo $var

var now has this value

==> echo ${var:-"this default value will not be displayed"}

var now has this value

==> echo ${var:+"this default value will be displayed"}

this default value will be displayed

==> echo ${var:?}var now has this value

==> echo ${var:?"this error message will not be sent to stderr"}

var now has this value

==> echo $?

0

﻿==> echo ${var:="this default value will not be displayed"}

var now has this value

==> echo $var

var now has this value

==> \_

* + ${var:offset:length} - substring substitution, with an optional length
  + positive offset (from beginning of string) for first character is 0
  + negative offset (from end of string) for last character is -1
  + positive length means length of substring
  + negative length is offset from end of string
  + here are some examples of substring substitution:

==> var1=abcdefghijklmnopqrstuvwxyz

==> echo ${var1:9}

jklmnopqrstuvwxyz

==> echo ${var1:9:5}

jklmn

==> echo ${var1:9:-5}

jklmnopqrstu

==> echo ${var1:-9}

# misinterpreted as a ":-" operator

﻿abcdefghijklmnopqrstuvwxyz

﻿==> echo ${var1: -9}

rstuvwxyz

﻿==> echo ${var1: -9:4}

rstu

==> echo ${var1: -9:-4}

rstuv

==> set abcdefghijklmnopqrstuvwxyz

==> echo ${1: -9:-4}

rstuv

==> num1=14; num2=7

﻿==> echo ${1: $((num1-5)): $((num2-2))}

jklmn

==> \_

* + ${#var} - gives the length of "$var"
  + ${!var\*} - displays all variable names beginning with "var" as a single string
  + ${!var@} - displays all variable names beginning with "var" as separate strings
  + here are some examples:

==> var1=lion

==> var2=tiger

==> var3="bear, oh my"

==> echo ${#var3}

11

==> echo ${!var\*}

﻿var1 var2 var3

==> \_

* + ${var%pattern} - removes the shortest part of "$var" that matches "pattern", from the end
  + ${var%%pattern} - removes the longest part of "$var" that matches "pattern", from the end
  + ${var#pattern} - removes the shortest part of "$var" that matches "pattern", from the beginning
  + ${var##pattern} - removes the longest part of "$var" that matches "pattern", from the beginning
  + if variable is "\*" or "@", the action is applied to each positional parameter
  + useful for string manipulation, for example directory processing:

==> x=$PWD

﻿==> echo $x

/home/lczegel/test/dir1

==> echo ${#x}

23

==> echo ${x%/\*}

/home/lczegel/tes﻿t

﻿==> echo ${x%%/\*}

==> echo ${x#\*/}

home/lczegel/test/dir1

==> echo ${x##\*/}

﻿dir1

﻿==> \_

* + here is an example of setting the primary prompt:

==> PS1='$PWD: '/home/lczegel/test/dir1: cd/home/lczegel: cd -/home/lczegel/test/dir1/home/lczegel/test/dir1: PS1='${PWD#~/}: 'test/dir1: cd/home/lczegel: cd -/home/lczegel/test/dir1test/dir1: \_

* + ${var/pattern/string} - replaces the longest part of "$var" that matches "pattern", with "string"
  + defaults to replacing the first match
  + if pattern begins with /, all matches will be replaced
  + if pattern begins with #, the match must be at the beginning of "$var"
  + if pattern begins with %, the match must be at the end of "$var"
  + if variable is "\*" or "@", the action is applied to each positional parameter
  + useful for simple string manipulation:

==> var1="My SCHOOL is seneca college"

﻿==> echo ${var1/seneca/the best}

My SCHOOL is the best college

==> echo ${var1/e/a}

My SCHOOL is saneca college

==> echo ${var1//e/a}

﻿My SCHOOL is sanaca collagadir1

﻿==> \_

* + ${var^pattern} - converts the first character of substrings that match "pattern" to uppercase
  + ${var^^pattern} - converts all characters of substrings that match "pattern" to uppercase
  + ${var,pattern} - converts the first character of substrings that match "pattern" to lowercase
  + ${var,,pattern} - converts all characters of substrings that match "pattern" to lowercase
  + if pattern is missing, default is "?", matching all characters in "$var"
  + if variable is "\*" or "@", the action is applied to each positional parameter
  + useful for simple string manipulation:

==> echo $var1My

SCHOOL is seneca college

==> echo ${var1^}

My SCHOOL is seneca college

==> echo ${var1^^}

MY SCHOOL IS SENECA COLLEGE

==> echo ${var1,}

my SCHOOL is seneca college

==> echo ${var1,,}

my school is seneca college

==> echo ${var1^^e}

My SCHOOL is sEnEca collEgE

==> echo ${var1,,O}

My SCHooL is seneca college

==> \_

**Shell Expansion**

* + the shell performs eight kinds of expansion, in the following order:
  + brace expansion
  + tilde expansion
  + variable expansion, command substitution, arithmetic expansion
  + process substitution
  + word splitting
  + pathname expansion

**Brace Expansion**

* + can generate sequences of character strings
  + similar idea to pathname expansion, without the need to match existing filenames
  + can use a comma-separated list, or a sequence
  + will not be expanded within quotes
  + here are some examples of brace expansion:

==> echo {1,2,3,5,8,13,21}

1 2 3 5 8 13 21

﻿==> echo {1..10}

1 2 3 4 5 6 7 8 9 10

﻿==> echo {01..10}

01 02 03 04 05 06 07 08 09 10

﻿==> echo {1..10..2}

1 3 5 7 9==> echo {10..1}

10 9 8 7 6 5 4 3 2 1

﻿==> echo {10..01..-2}

10 08 06 04 02

﻿==> echo {a..t..2}

a c e g i k m o q s

==> \_

* + here are some examples using brace expansion to generate filenames:

==> touch file{1..10..2}.txt

==> ls

file1.txt file3.txt file5.txt file7.txt file9.txt

==> rm file{0..10..3}.\*

rm: cannot remove `file0.\*': No such file or directoryrm: cannot remove `file6.\*': No such file or directory

==> ls

file1.txt file5.txt file7.txt

==> rm \*

==> touch pic{0..10..3}.{jpg,gif,jpeg}

==> ls

pic0.gif pic0.jpg pic3.jpeg pic6.gif pic6.jpg pic9.jpegpic0.jpeg pic3.gif pic3.jpg pic6.jpeg pic9.gif pic9.jpg

==> mkdir dir{1..5}

==> touch dir{1..5}/file{01..10}

==> ls dir3

file01 file02 file03 file04 file05 file06 file07 file08 file09 file10

==> \_

**Tilde Expansion**

* + can be used as shortcut to some directory names
  + ~ is home directory of current user
  + ~username is home directory of specified user
  + ~+ is the current directory, same as $PWD
  + ~- is the previous directory, same as $OLDPWD
  + will not be expanded within quotes
  + here are some examples of tilde expansion:

==> echo ~/home/lczegel

==> echo "~"~

==> index=~/public\_html/index.html

==> ls -l $index-rw-r--r-- 1 lczegel users 596 May 24 2013 /home/lczegel/public\_html/index.html

==> touch $index

﻿==> ls -l

$index-rw-r--r-- 1 lczegel users 596 Apr 7 22:50 /home/lczegel/public\_html/index.html

==> echo $PWD

﻿/home/lczegel/test/dir1

==> echo ~+

/home/lczegel/test/dir1

==> echo $OLDPWD

﻿/home/lczegel

==> echo ~-

/home/lczegel

==> file=~john.selmys/public/file16

==> echo $file

﻿/home/john.selmys/public/file16

==> \_

* + can be used to find files within the directory stack
  + dirs - displays the directory stack
  + pushd dirname - pushes "dirname" onto directory stack and changes to it
  + popd - pops top directory name from directory stack and changes to next one
  + cd dirname - replaces top of directory stack with "dirname" and changes to it
  + the top of the directory stack is always the current directory
  + ~3 or ~+3 is replaced by the 3rd name from the top of the directory stack, starting at 0
  + ~-3 is replaced by the 3rd name from the bottom of the directory stack, starting at 0
  + here are some examples of manipulating the directory stack:

/home/lczegel

﻿: dirs~/home/lczegel

﻿: cd test/dir1~/test/dir1

﻿: dirs~/test/dir1~/test/dir1

﻿: pushd ~~ ~/test/dir1/home/lczegel

﻿: pushd /etc/etc ~ ~/test/dir1/etc

﻿: pushd /var/var /etc ~ ~/test/dir1/var

﻿: cd/home/lczegel

﻿: dirs~ /etc ~ ~/test/dir1/home/lczegel

﻿: echo ~3/home/lczegel/test/dir1/home/lczegel

﻿: echo ~-2/etc/home/lczegel

﻿: dirs~ /etc ~ ~/test/dir1/home/lczegel

﻿: popd/etc ~ ~/test/dir1/etc

﻿: popd~ ~/test/dir1/home/lczegel

﻿: popd~/test/dir1~/test/dir1

﻿: popd-bash

﻿: popd

﻿: directory stack empty~/test/dir1

﻿: \_

**Command Substitution**

* + $(command-line) - new style (korn, bash) of command substitution, easily nested to multiple levels
  + `command-line` - old style (bourne) of command substitution, nesting requires escaping inner back-quotes, only one level of nesting possible
  + examples of nested command substitution:

==> echo -e "One year from today:\n\n $(cal $(date +'%m %Y' | awk '{print $1, $2 + 1}'))"One year from today:June 2018

﻿Su Mo Tu We Th Fr Sa1 23 4 5 6 7 8 910 11 12 13 14 15 1617 18 19 20 21 22 2324 25 26 27 28 29 30

﻿==> echo -e "One year from today:\n\n `cal \`date +'%m %Y' | awk '{print $1, $2 + 1}'\``"One year from today:June 2018

﻿Su Mo Tu We Th Fr Sa

1 23 4 5 6 7 8 9

10 11 12 13 14 15 16

17 18 19 20 21 22 23

﻿24 25 26 27 28 29 30

﻿==> \_

**Arithmetic Expansion**

* + $(( expression )) - new style (korn, bash) of arithmetic expansion, easily nested
  + an example of arithmetic expansion:

==> echo -e "One year from today:\n\n $(cal $(date +'%m') $(($(date +'%Y') + 1)))"One year from today:June 2018

﻿Su Mo Tu We Th Fr Sa

1 23 4 5 6 7 8 9

﻿10 11 12 13 14 15 16

﻿17 18 19 20 21 22 23

﻿24 25 26 27 28 29 30

﻿==> \_

**Word Splitting**

* + shell splits results of previous expansions, that were not in quotes, into words
  + examples of word splitting:

==> var1=ford

==> grep $var1 cars

ford mustang 65 45 17000

﻿ford ltd 83 15 10500

﻿ford thundbd 84 10 17000

﻿ford bronco 83 25 9525

﻿==> var1="ford ltd"

﻿==> grep $var1 cars

grep: ltd: No such file or directory

cars:ford mustang 65 45 17000

cars:ford ltd 83 15 10500

cars:ford thundbd 84 10 17000

﻿cars:ford bronco 83 25 9525

﻿==> grep "$var1" cars

ford ltd 83 15 10500

==> \_

**Shell Startup**

**bash startup**

* + /etc/profile executes, then .bash\_profile in home directory
  + .bash\_login is used if .bash\_profile doesn't exist
  + .profile is used if .bash\_profile and .bash\_login don't exist
  + .bashrc is used for interactive sub-shells

**ksh startup**

* + /etc/profile executes, then .profile in home directory
  + .kshrc is used for interactive sub-shells

**alias**

* + assigns a new name to an existing utility
  + eg. alias dir=ls
  + eg. alias ls='ls -al'
  + can be useful, but can also make your scripts cryptic

**Odds and Ends**

**Some Useful Commands and Options**

* + nl - number lines of a file, lots of options for formatting the line numbers:

==> cat cars

plym fury 77 73 2500

chevy nova 79 60 3000

ford mustang 65 45 17000

volvo gl 78 102 9850

ford ltd 83 15 10500

Chevy nova 80 50 3500

fiat 600 65 115 450

honda accord 81 30 6000

ford thundbd 84 10 17000

toyota tercel 82 180 750

chevy impala 65 85 1550

ford bronco 83 25 9525

==> nl cars1

plym fury 77 73 25002

chevy nova 79 60 30003

ford mustang 65 45 170004

volvo gl 78 102 98505

ford ltd 83 15 105006

Chevy nova 80 50 35007

fiat 600 65 115 4508

honda accord 81 30 60009

ford thundbd 84 10 1700010

toyota tercel 82 180 75011

chevy impala 65 85 155012

ford bronco 83 25 9525

==> \_

* + note that nl results can be duplicated using commands already discussed
  + click here for one possible solution, but try it yourself first
  + cat - has some options similar to nl, but with less flexible formatting:

==> cat cars.blanklines

plym fury 77 73 2500

chevy nova 79 60 3000

ford mustang 65 45 17000

volvo gl 78 102 9850

ford ltd 83 15 10500

Chevy nova 80 50 3500

fiat 600 65 115 450

honda accord 81 30 6000

ford thundbd 84 10 17000

toyota tercel 82 180 750

chevy impala 65 85 1550

ford bronco 83 25 9525

==> cat -n cars.blanklines1

plym fury 77 73 25002

chevy nova 79 60 3000345

ford mustang 65 45 170006

volvo gl 78 102 98507

ford ltd 83 15 105008

Chevy nova 80 50 35009

fiat 600 65 115 450101112

honda accord 81 30 600013

ford thundbd 84 10 1700014

toyota tercel 82 180 75015

chevy impala 65 85 155016

ford bronco 83 25 9525

==> cat -b cars.blanklines1

plym fury 77 73 25002

chevy nova 79 60 30003

ford mustang 65 45 170004

volvo gl 78 102 98505

ford ltd 83 15 105006

Chevy nova 80 50 35007

fiat 600 65 115 4508

honda accord 81 30 60009

ford thundbd 84 10 1700010

toyota tercel 82 180 75011

chevy impala 65 85 155012

ford bronco 83 25 9525

==> \_

* + cat - has an option to display special characters, for example tab displays as ^I, newline displays as $:

==> cat cars.tab

plym fury 77 73 2500

chevy nova 79 60 3000

ford mustang 65 45 17000

volvo gl 78 102 9850

ford ltd 83 15 10500

Chevy nova 80 50 3500

fiat 600 65 115 450

honda accord 81 30 6000

ford thundbd 84 10 17000

toyota tercel 82 180 750

chevy impala 65 85 1550

ford bronco 83 25 9525

﻿==> cat -A cars.tab

plym^Ifury^I77^I73^I2500$

﻿chevy^Inova^I79^I60^I3000$

﻿ford^Imustang^I65^I45^I17000$

﻿volvo^Igl^I78^I102^I9850$

﻿ford^Iltd^I83^I15^I10500$

Chevy^Inova^I80^I50^I3500$

﻿fiat^I600^I65^I115^I450$

﻿honda^Iaccord^I81^I30^I6000$

﻿ford^Ithundbd^I84^I10^I17000$

﻿toyota^Itercel^I82^I180^I750$

﻿chevy^Iimpala^I65^I85^I1550$

﻿ford^Ibronco^I83^I25^I9525$

﻿==> \_

* + tac - display lines of a file backwards, last-line first:

==> cat cars

plym fury 77 73 2500

chevy nova 79 60 3000

ford mustang 65 45 17000

volvo gl 78 102 9850

ford ltd 83 15 10500

Chevy nova 80 50 3500

fiat 600 65 115 450

honda accord 81 30 6000

ford thundbd 84 10 17000

toyota tercel 82 180 750

chevy impala 65 85 1550

ford bronco 83 25 9525

==> tac cars

ford bronco 83 25 9525

chevy impala 65 85 1550

toyota tercel 82 180 750

ford thundbd 84 10 17000

honda accord 81 30 6000

fiat 600 65 115 450

Chevy nova 80 50 3500

ford ltd 83 15 10500

volvo gl 78 102 9850

ford mustang 65 45 17000

chevy nova 79 60 3000

plym fury 77 73 2500

==> \_

* + tac results can be duplicated using commands already discussed
  + click here for one possible solution, but try it yourself first
  + here is an example of displaying the fields in each line backwards, last-field first:

==> cat cars

plym fury 77 73 2500

chevy nova 79 60 3000

ford mustang 65 45 17000

volvo gl 78 102 9850

ford ltd 83 15 10500

Chevy nova 80 50 3500

fiat 600 65 115 450

honda accord 81 30 6000

ford thundbd 84 10 17000

toyota tercel 82 180 750

chevy impala 65 85 1550

ford bronco 83 25 9525

﻿==> awk '{ for (i=NF; i>0; i--) printf("%-8s", $i); printf("\n") }' cars

2500 73 77 fury plym

3000 60 79 nova chevy

17000 45 65 mustang ford

9850 102 78 gl volvo

10500 15 83 ltd ford

3500 50 80 nova Chevy

﻿450 115 65 600 fiat

6000 30 81 accord honda

17000 10 84 thundbd ford

750 180 82 tercel toyota

1550 85 65 impala chevy

9525 25 83 bronco ford

==> \_

* + here is an example of a similar awk loop, working with numeric files:

==> cat numbers

3 5 3 7 9 6 4 2 61.96 2.95 1.32 5.49 17.543 1.96 5 2.95 3 1.32 7 5.49 9 17.54 6 4 2 6

﻿==> awk '{ s=0; for (i=1; i<=NF; i++) s+=$i; t+=s; print "Subtotal Line " NR ": " s }> END { print "Total: " t }' numbers

Subtotal Line 1: 45Subtotal Line 2: 29.26Subtotal Line 3: 74.26Total: 148.52

﻿==> \_

* + bc - a calculator, scale is number of decimal places:

==> degrees\_c=21

﻿==> echo "scale = 2; $degrees\_c \* 9 / 5 + 32" | bc

69.80

﻿==> echo "$degrees\_c celcius is $(echo 'scale = 2; '$degrees\_c' \* 9 / 5 + 32' | bc) fahrenheit"

﻿21 celcius is 69.80 fahrenheit

==> set 80==> echo "scale = 2; ($1 - 32) \* 5 / 9" | bc

26.66

﻿==> echo "$1 fahrenheit is $(echo 'scale = 2; ('$1' - 32) \* 5 / 9' | bc) celcius"

﻿80 fahrenheit is 26.66 celcius

==> \_

* + bc results can be duplicated using commands already discussed
  + click here for one possible solution, but try it yourself first
  + ls -S - sorts by file size, largest to smallest, -r reverses the order:

==> ls -l

total 24

-r-------- 1 lczegel users 445 Jun 16 17:59 cars

-r-------- 1 lczegel users 449 Jun 16 17:59 cars.blanklines

-r-------- 1 lczegel users 273 Jun 16 17:59 cars.tab

-rwx------ 1 lczegel users 273 May 27 03:46 trapsig

-rw------- 1 lczegel users 533 Jun 16 17:59 words

-rwx------ 1 lczegel users 482 May 27 03:23 yyy

-rw------- 1 lczegel users 0 May 29 15:58 zzz

==> ls -lr

total 24-rw------- 1 lczegel users 0 May 29 15:58 zzz

-rwx------ 1 lczegel users 482 May 27 03:23 yyy

-rw------- 1 lczegel users 533 Jun 16 17:59 words

-rwx------ 1 lczegel users 273 May 27 03:46 trapsig

-r-------- 1 lczegel users 273 Jun 16 17:59 cars.tab

-r-------- 1 lczegel users 449 Jun 16 17:59 cars.blanklines

-r-------- 1 lczegel users 445 Jun 16 17:59 cars

==> ls -lStotal 24-rw------- 1 lczegel users 533 Jun 16 17:59 words

-rwx------ 1 lczegel users 482 May 27 03:23 yyy

-r-------- 1 lczegel users 449 Jun 16 17:59 cars.blanklines

-r-------- 1 lczegel users 445 Jun 16 17:59 cars

-r-------- 1 lczegel users 273 Jun 16 17:59 cars.tab-rwx------ 1 lczegel users 273 May 27 03:46 trapsig-rw------- 1 lczegel users 0 May 29 15:58 zzz==> ls -Srzzz trapsig cars.tab cars cars.blanklines yyy words==> \_

* + ls -S results can be duplicated using commands already discussed
  + click here for one possible solution, but try it yourself first
  + seq - generates sequences of numbers:

==> seq 512345==> seq 1 2 1013579==> seq 1.3 1.5 101.32.84.35.87.38.8==> seq 6 -1.5 -36.04.53.01.50.0-1.5-3.0==> for x in $(seq 1 0.1 2); do echo $x; done1.01.11.21.31.41.51.61.71.81.92.0==> \_

* + $RANDOM - generates a pseudo-random number between 0 and 32767:

==> echo $RANDOM23218==> die1=$((RANDOM % 6 + 1)); die2=$((RANDOM % 6 + 1)); echo "$die1 + $die2 = $((die1 + die2))"2 + 1 = 3==> die1=$((RANDOM % 6 + 1)); die2=$((RANDOM % 6 + 1)); echo "$die1 + $die2 = $((die1 + die2))"5 + 1 = 6==> die1=$((RANDOM % 6 + 1)); die2=$((RANDOM % 6 + 1)); echo "$die1 + $die2 = $((die1 + die2))"6 + 3 = 9==> die1=$((RANDOM % 6 + 1)); die2=$((RANDOM % 6 + 1)); echo "$die1 + $die2 = $((die1 + die2))"5 + 4 = 9==> die1=$((RANDOM % 6 + 1)); die2=$((RANDOM % 6 + 1)); echo "$die1 + $die2 = $((die1 + die2))"3 + 2 = 5==> \_

* + fold and fmt - format text, each with lots of options:

==> cat wordsHere are a bunch of words used to demonstrate variouscommands. Some of the lines are quite short and some are quite long. However, they will all beused by all the commands. And I've completely run out of thingsto say, so maybe I'll just repeat this paragraph.Here are a bunch of words used to demonstrate variouscommands. Some of the lines are quite short and some are quite long. However, they will all beused by all the commands. And I've completely run out of thingsto say, so maybe I'll just repeat this paragraph.==> fold -w80 wordsHere are a bunch of words used to demonstrate variouscommands. Some of the lines are quite short and some are quite long. However,they will all beused by all the commands. And I've completely run out of thingsto say, so maybe I'll just repeat this paragraph.Here are a bunch of words used to demonstrate variouscommands. Some of the lines are quite short and some are quite long. However,they will all beused by all the commands. And I've completely run out of thingsto say, so maybe I'll just repeat this paragraph.==> fmt -w80 wordsHere are a bunch of words used to demonstrate various commands. Some ofthe lines are quite short and some are quite long. However, they will allbe used by all the commands. And I've completely run out of things to say,so maybe I'll just repeat this paragraph.Here are a bunch of words used to demonstrate various commands. Some ofthe lines are quite short and some are quite long. However, they will allbe used by all the commands. And I've completely run out of things to say,so maybe I'll just repeat this paragraph.==> \_

* + cut --complement - invert cut field or character selection:

==> cat carsplym fury 77 73 2500chevy nova 79 60 3000ford mustang 65 45 17000volvo gl 78 102 9850ford ltd 83 15 10500Chevy nova 80 50 3500fiat 600 65 115 450honda accord 81 30 6000ford thundbd 84 10 17000toyota tercel 82 180 750chevy impala 65 85 1550ford bronco 83 25 9525==> tr -s ' ' < cars | cut -d' ' -f3,477 7379 6065 4578 10283 1580 5065 11581 3084 1082 18065 8583 25==> tr -s ' ' < cars | cut -d' ' -f3,4 --complementplym fury 2500chevy nova 3000ford mustang 17000volvo gl 9850ford ltd 10500Chevy nova 3500fiat 600 450honda accord 6000ford thundbd 17000toyota tercel 750chevy impala 1550ford bronco 9525==> \_

* + here are some examples of using sed to add or replace lines in a file:

==> cat carsplym fury 77 73 2500chevy nova 79 60 3000ford mustang 65 45 17000volvo gl 78 102 9850ford ltd 83 15 10500Chevy nova 80 50 3500fiat 600 65 115 450honda accord 81 30 6000ford thundbd 84 10 17000toyota tercel 82 180 750chevy impala 65 85 1550ford bronco 83 25 9525==> sed '3 aThis is a new line after the 3rd line' carsplym fury 77 73 2500chevy nova 79 60 3000ford mustang 65 45 17000This is a new line after the 3rd linevolvo gl 78 102 9850ford ltd 83 15 10500Chevy nova 80 50 3500fiat 600 65 115 450honda accord 81 30 6000ford thundbd 84 10 17000toyota tercel 82 180 750chevy impala 65 85 1550ford bronco 83 25 9525==> sed '/ford/ aThis is a new line after each ford' carsplym fury 77 73 2500chevy nova 79 60 3000ford mustang 65 45 17000This is a new line after each fordvolvo gl 78 102 9850ford ltd 83 15 10500This is a new line after each fordChevy nova 80 50 3500fiat 600 65 115 450honda accord 81 30 6000ford thundbd 84 10 17000This is a new line after each fordtoyota tercel 82 180 750chevy impala 65 85 1550ford bronco 83 25 9525This is a new line after each ford==> sed '/chevy/ cThis is a new line replacing each chevy' carsplym fury 77 73 2500This is a new line replacing each chevyford mustang 65 45 17000volvo gl 78 102 9850ford ltd 83 15 10500Chevy nova 80 50 3500fiat 600 65 115 450honda accord 81 30 6000ford thundbd 84 10 17000toyota tercel 82 180 750This is a new line replacing each chevyford bronco 83 25 9525==> \_