module1exercise

================================================

1. create a symbolic link (why not a hard link?)

ln -s /home/distribution/cnguyen/cis18b link18b

2. create a variable to store the path and store the variable in .bashrc

var18b=/home/distribution/cnguyen/cis18b

================================================

use vim to add one more class to module1input: pe 71 with 1 unit

vim module1input (the cursor should be at line 1)

G (go to the last line)

O (open a new line above the current line

and go in insert mode)

pe,71,1 (type the text)

escape (get back to command mode)

:wq (save and quit)

=======================================================

tr 'a-z,' 'A-Z ' < module1input | tee m1

tr 'a-z,' 'A-Z ' < module1input | tee m1 > m2

means that the first copy of the output will go to m1, and then the; 2nd copy will be redirected to file m2

=======================================================

show the long listing of "m1"

ls -l m1

what attributes do you see for "m1"?

first character: file type, in this case a regular file

next 9 characters: permission for user, then group, then others

next column: number of hard links, in this case 1 hard link

next column: owner id

next column: owner group

next column: file size in bytes

next columns: access date and time

last column: filename

=======================================================

without using vim, append to "m1" 2 lines

line 1: "classes" (with quotes)

line 2: your terminal id

(echo \"classes\" ; tty) >> m1 (2 lines)

(echo \"classes\ `tty`\") >> m1 (1 line)

=======================================================

create a new module1input file, with the lines 2-7 of file "m1"

head -7 m1| tail -6 > module1input

=======================================================

show number of lines in the file "m1"

wc -l m1 | cut -d' ' -f1

=======================================================

show on screen the lines that are 5 units, with class names sorted

in revese order

egrep '[^.]5$' module1input | sort -r

the regex is for: a character that's not a period, followed by

a character 5, followed by the end of the line

we need this specific regex so the 4.5's on the line won't

pass the filter

sort -r is to do reverse sort

the regex can also be: ' 5$'

which means: a space, followed by a 5, followed by end of line

------------------------------------

egrep

------------------------------------

show the filenames of files in the cis18b directory that have

punctuation marks in the file

[sehun $] egrep -l '[[:punct:]]' \*

clean.sed

clean.sh

------------------------------------

show the count of the lines that have punctuation marks in each

of the files of the cis18b directory

[sehun $] egrep -c '[[:punct:]]' \*

clean.sed:12

clean.sh:20

lab1-2in:50

------------------------------------

show the line number and the lines that have only punctuation marks

(and nothing else on the line) in the files of the cis18b directory

[sehun $] egrep -n '^[[:punct:]]+$' \*

clean.sh:5:#

clean.sh:11:#

lab3in:11:===========================

------------------------------------

show the line number and the lines that don't contain any number

in the file module1input

[sehun $] egrep -nv '[[:digit:]]' module1input

1:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

7:\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

------------------------------------

show the line number of all lines that have only whitespace characters, for all files in the cis18b directory

[sehun $] egrep -n '^[[:space:]]+$' \*

module2exerciseSolution:6:

module2exerciseSolution:13:

module2exerciseSolution:18:

------------------------------------

go to your home directory

create 2 new files fA and fB

create a symbolic link to file fA

create a directory d2

touch fA fB

ln -s fA linkA ## note that the path stored in linkA is: fA

mkdir d2

------------------------------------

start from your home directory, find all links to file fA

find ~ -type l

this gives us you all the links,

regardless of whether a link points to fA or not

find ~ -links +1

this gives us all files with more than 1 hard links

find ~ -lname '\*/fA'

this doesn't give us any file match, because we tell find

that the link must have a path to fA that matches: \*/fA

But our path for linkA is just "fA"

Therefore we ned to use: find ~ -lname '\*fA'

Explanation: -lname looks inside each link to see if the path stored in the link ends with fA, indicating that the link

points to fA. We need to include the \* in front of fA because the path to fA could contain multiple directory names, such as ~/myDir/fA or it could simply be fA as we have in the example linkA.

------------------------------------

start from your home directory, find all directories that have

'd' in the name

[ohericcc@voyager ~]$ find -type d -name '\*d\*'

./module7exerciseSolution

./TameImpala/dir1

./mod6

Note: notice the difference in the output path of the command above,

vs. if you are at the home directory and type:

[ohericcc@voyager ~]$ find ~ -type d -name '\*d\*'

/home/student/ohericcc/module7exerciseSolution

/home/student/ohericcc/TameImpala/dir1

/home/student/ohericcc/mod6

------------------------------------

start from your home directory, find all regular files that have

a size less than 1K bytes

find ~ -type f -size -1k

------------------------------------

start from your home directory, delete all regular files that are empty

Using -exec option:

find ~ -type f -empty -exec rm -i {} \;

note the difference between using exec and xarg:

find ~ -type f -empty | xargs rm -i

Explanation for the 2 different output:

xargs processes all the output of find as one block so we don't

have a chance to respond to the -i option

exec processes each of the output file of find one at a time as find

'finds' them so we have a chance to respond to the -i option

In general, when you have a few files to process, then -exec option

is better because it allows the interactive option

when you have lots of files to process, then xargs is better because

it's more efficient and can handle a long list of filenames, whereas

-exec may run out of memory buffer it the list is too long

------------------------------------

start from your home directory, do the long listing of all files

that are regular files

find ~ -type f -exec ls -l {} \;

or

find ~ -type f | xargs ls -l

### extra credit question 1:

### in class we also ran: find ~ -type f | ls -l

### and saw a different output

### Explain the difference in the output of:

find ~ -type f | xargs ls -l

### Answer to EC 1:

The output of find is piped to xargs as a list, xargs goes through

each file in the list and runs ls -l with the filename

### and the output of:

find ~ -type f | ls -l

### Answer to EC 1:

The output of find is piped to ls -l, and they get ignored

and ls -l runs by itself

Why did the piped output get ignored?

ls -l doesn't use standard input, therefore it won't accept

anything from a pipe

------------------------------------

start from your home directory, update the modified time of all

regular files that have modified time of less than 1 day

### extra credit question 2:

### find (pun intended) the solution to this question

### and test it out before turning it in

find ~ -type f -mtime -1 | xargs touch

------------------------------------

----------------------------------

sed

----------------------------------

Run the print command of sed with the module3Ainput

Should you use the command line format or the sed script format?

Answer: Use command line format because it is one short instruction

sed 'p' module3Ainput

----------------------------------

How do you not get duplicate lines in the output when using the

print instruction?

sed -n 'p' module3Ainput

----------------------------------

Print a line number and each line of module3Ainput

sed '=' module3Ainput

----------------------------------

Add a line with the word NEW, before all lines that has the word 'line'

sed '/line/ i NEW' module3Ainput

----------------------------------

Add the word NEW to the front of all lines that has the word 'line'

sed 's/.\*line.\*/NEW &/' module3Ainput

----------------------------------

Change the last line so it says: This is 10

sed '$c This is 10' module3Ainput

----------------------------------

Delete all lines that have a number

sed '/[0-9]/d' module3Ainput

----------------------------------

Delete all lines that don't have a number

sed '/[0-9]/! d' module3Ainput

----------------------------------

Change all words 'line' to 'LINE' and show all lines

sed 's/line/LINE/g' module3Ainput

----------------------------------

Add \* in front and back of all words 'line' but show only the lines

that are changed

sed -n 's/line/\*&\*/gp' module3Ainput

----------------------------------

Create a new file with module1input inserted between lines 3 and 4 of

module3Ainput

sed '3r module1input' module3Ainput > newfile

Extra discussion:

In class we also tried the w instruction:

sed '3r module1input ; w newfile' module3Ainput

but this doesn't work. The reason is that the r instruction looks at

the rest of the sed instruction as the filename to be read in, so the

file name it's looking for is: "module1input ; w newfile"

which doesn't exist.

To use both the r and w command for this, we need to write a sed script

which will have 2 lines:

3r module1input

w newfile

and sed scripts are covered next week.

---------------------------------

Print all lines up to the first line with the word 'Line'

sed '/Line/q' module3Ainput (using the q command)

----------------------------------

Extra challenge:

Print the first 6 lines, with all spaces converted to -

in each of the first 4 lines only

sed '1,4 s/ /-/g; 6q' module3Ainput

----------------------------------

For sed scripts below

dept;class;units;time;day

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

morning

cis;18b;4.5;11:30;TTh

chem;50;5;9:30;MWF

ewrt;1a;5;9:30;MWF

acct;1a;5;8:30;MTWTh

afternoon

math;10;5;1:30;MW

cis;22a;4.5;1:30;TTh

pe;50;1;2:00;F

spch;10;4;6:30;TTh

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

----------------------------------

# print every 2 lines, with a line of +++++ before and after each

# pair of lines to run: sed -f s1 module3Binput

i +++++ # first, insert ++++ above current line

N # then bring in 2nd line

----------------------------------

# print every third line

# to run: sed -nf s2 module3Binput

# n; n; p

----------------------------------

print all afternoon class information: in column format (with tab

delimiter), with one line per class, with +++++ before the first class

and after the last class

#/afternoon/,$ { s/afternoon/++++/

s/\\*+/++++/ # change ; to tab

y/;/\t/

p

}

--------------------------------------------

print the day and time of CIS classes, with a heading of

"CIS classes day and time" and with data in column format

1i CIS classes day and time

/cis/ s/;/\t/gp

------------------ 5 --------------------------

create a new module3out file that has this information

dept class units time day

===================================

AM

cis 18b 4.5 11:30 TTh

chem 50 5 9:30 MWF

ewrt 1a 5 9:30 MWF

acct 1a 5 8:30 MTWTh

PM

math 10 5 1:30 MW

cis 22b 4.5 1:30 TTh

pe 50 1 2:00 F

spch 10 4 6:30 TTh

===================================

---------------- 6 -------------------------

instead of creating a new module3out file in the question above,

change the module3Binput file itself

# to run for exercise 5: sed -f s5 module3Binput

# to run for exercise 6: sed -i -f s5 module3Binput

y/;/\t/

s/\\*/==/g

s/morning/AM/

/afternoon/ c PM

# or, to get w to send the line out to file, we change c to s:

s/afternoon/PM/

/acct/ a

w module3out

-------------------------------

awk

-------------------------------

show lines 5 and 8 of the output of who, with a tab separating the line number from the line of output

who | awk 'NR == 5 || NR == 8 {print NR, "\t", $0}'

------------------------------

for all users currently logged in, show user ids that

have digits or punctuation marks in them

who | awk '$1 ~ /[0-9[:punct:]]/ {print $1}' OR

who | awk '$1 ~ /[[:digit:][:punct:]]/ {print $1}'

-------------------------------------------

print the lines of code of clean.sh in reverse (last line first, first line last). Lines of code means no comment lines and no blank lines

{

arr[NR] = $0

}

END {

for(i=NR; i >=1; i--) {

if(arr[i] !~ /^#/ && arr[i] !~ /^$/)

print arr[i]

}

}

------------------------------

print the lines of module4AInput, but with ; replaced

by tab

BEGIN { FS=";" }

{

for(i=1;i<=NF;i++) {

printf("%s\t",$i)

}

printf("\n")

}

------------------------------

print the characters that are present in line 3 of clean.sh, and, next to each letter, print a count of how many of these letters are in the lines.

if(NR == 3) {

num = split($0,arr,"")

for(i=1;i<=num;i++)

charArr[arr[i]]++

}

}

END {

for(x in charArr)

print x, charArr[x]

}

------------------------------

for the current directory, print the access date and time and name of the file with the oldest access time

# awk script for module 4B exercise, question 5

# to run: ls -l | awk -f a5

# this script uses split to separate the $8 field (hh:mm) into hour and minute

# This is the "safest" way to solve the problem if you

# are not sure that the time format is always a "nice" hh:mm such as 09:50, instead of just 9:50

BEGIN {

hours = 24 # init hours and mins to largest values

mins = 60

}

# skip the first line, which has no $8 so it doesn't #cause the first $8 to be 0, thus defeating our

# initializing the hours and mins to the max values above

NR != 1 {

split($8, arr, ":") # hh:mm becomes arr[1] = hh

# arr[2] = mm

if (arr[1] < hours) # if current hh < hours

{

hours = arr[1] # then it's the earliest time

mins = arr[2] # and we record all values

date = $6" "$7

name = $9

}

else if (arr[1] == hours) # if current hh == hours

{

if (arr[2] < mins) # then check if current mm < mins

{

mins = arr[2] # if it is, then it's the earliest time,

date = $6" "$7 # record all values

name = $9

}

}

}

END { print name, "at", date, hours":"mins }

# last field to be printed is a string that we form from 3 strings

-------------------------------

make

------------------------------

go to printing.cpp and change the function printHello to

print Hi instead of Hello

sed -i 's/Hello,/Hi/' printing.cpp

reminder: -i option to overwrite the original file

-------------------------------------------------

run the executable. Is the Hello changed to Hi? Why not?

Hello is still printed because the executable is not updated or not rebuilt

-------------------------------------------------

What steps do you need to do to get a new executable?

Do the steps to see the change in the program output

steps:

1. recompile printing.cpp to produce a new printing.o file

g++ -c printing.cpp

2. re-link all the object files to produce a new executable

g++ -o hello hello.o printing.o name.o

-------------------------------------------------

basic makefile

-------------------------------------------------

create a simple makefile to do the same as the steps above pay attention to the order of the steps, so that you only have to type: make on the command line

# the first rule of the make file is the one that will #run when typing make on the command line.

# First Rule is to build executable:

myfile : hello.o printing.o name.o

g++ -o myfile hello.o printing.o name.o

# Rules to build object file from source files

name.o : name.cpp

g++ -c name.cpp

printing.o : printing.cpp

g++ -c printing.cpp

hello.o : hello.cpp

g++ -c hello.cpp

-------------------------------------------------

compile just name.cpp. Does it compile?

make name.o (output: name.o is up to date)

so it doesn't compile

What do you need to do first to get it to compile? Why?

need to remove name.o or touch name.cpp first because

- remove name.o forces make to rebuild name.o since the file doesn't exist

- touch name.cpp means the timestamp for name.c will be later

than the timestamp for name.o. This causes make to rebuild name.o

The important point is that make will only build when necessary.

It will not blindly recompile and relink everything

This saves a lot of time and work when you have a big project with

many many files

-------------------------------------------------

makefile with macros

-------------------------------------------------

add macros to make the makefile shorter and more flexible:

- macro for list of .o files

- macro for executable filename

- macro for the compiler name

# macros need to be at the top of the file so make can see them

# before you use them

OBJ = hello.o printing.o name.o

EXE = myfile

CC = g++

# Rule to build executable:

$(EXE) : $(OBJ)

$(CC) -o $(EXE) $(OBJ)

# Rules to build object file from source files

name.o : name.cpp

$(CC) -c name.cpp

printing.o : printing.cpp

$(CC) -c printing.cpp

hello.o : hello.cpp

$(CC) -c hello.cpp

-------------------------------------------------

rebuild the executable but change the executable name to

newExe

make EXE=newExe

having the macro EXE means that we can select a different executable filename for this particular run, instead of having to use the filename already in the makefile

-------------------------------------------------

makefile with phony targets

-------------------------------------------------

add a target that will do a long listing of all the files

in the current directory, call that target long

# macros need to be at the top

OBJ = hello.o printing.o name.o

EXE = myfile

CC = g++

# declare "long" to be a phony target (which is a label)

# so make doesn't search for a file called long

.PHONY : long

# Rule to build executable:

$(EXE) : $(OBJ)

$(CC) -o $(EXE) $(OBJ)

# Rules to build object file from source files

name.o : name.cpp

$(CC) -c name.cpp

printing.o : printing.cpp

$(CC) -c printing.cpp

hello.o : hello.cpp

$(CC) -c hello.cpp

long :

@echo Here is the long listing

@ls -l

# the @ means the command will not be echoed out to screen before it runs

-------------------------------------------------

run make to do a long listing of the current directory

make long

-------------------------------------------------

create a new, empty file called long

touch long

-------------------------------------------------

run make to do a long listing of the current directory

without the .PHONY declaration, make long will not run ls -l

-------------------------------------------------

have make not print the ls -l command when it runs

see last makefile

-------------------------------------------------

makefile with common suffix rule

-------------------------------------------------

use the common suffix rule to shorten the makefile

see next makefile

-------------------------------------------------

change the makefile so that when you type make list

it does a long listing of all the files in the current

directory and then build a new executable if necessary

see next makefile

# makefile for part 4 of module 5 exercise

# macros need to be at the top

OBJ = hello.o printing.o name.o

EXE = myfile

CC = g++

# declare "long" to be a phony targer (or a label)

.PHONY : long list

# Rule to build executable:

$(EXE) : $(OBJ)

$(CC) -o $(EXE) $(OBJ)

# Rules to build object file from source files, using common suffix

%.o : %.cpp

$(CC) -c $^

long :

@ls -l

======================

gzip/gunzip/tar

======================

create a compressed testfile, use the verbose option

gzip -v testfile

the -v option should tell you that testfile is replaced

with testfile.gz and show the compression rate as a percentage

====================

decompress the gzip file. Observe the access time of the

decompressed file

We tried both ways:

gzip -d testfile.gz

gunzip testfile.gz

the timestamp of testfile is still the same as the original testfile

-------------

Note that the .gz extension is required. We tried this test:

1. first remove the .gz extension

cp testfile.gz testfile

2. decompress the file

gunzip testfile

We got an error message from gzip that filename suffix is not correct We changed the filename back to .gz to run gunzip with it.

-------------

Since gzip removes the original file, someone asked in class about how to use the -c option and whether we need to name the output file ourselves, instead of depending on gzip as with default mode.

The -c option is useful when we want to keep the original file

gzip -c testfile > mytestfile.gz

gzip will compress the file but instead of automatically creating

a testfile.gz and deleting testfile, it will keep testfile and

send back the compressed data, which we redirect to mytestfile.gz

Now we have both testfile and mytestfile.gz in the directory

===================

copy the module5exercise directory to your local directory and give it the name testdir

create an archive file for the testdir directory, make sure it's not compressed

tar cvf testdir.tar testdir

there can be a - in front of cvf: -cvf

due to the -v option, you should see the list of files

that are being archived into testdir.tar

===================

append the testfile to the archive file

tar -rf testdir.tar testfile

===================

show the files that are in the archive file

tar -tf testdir.tar

===================

compress the archive file

gzip testdir.tar

ls \*tar\*

to see the zipped filename

===================

create a new directory called mod6

copy the archive file to mod6

move to the mod6 directory and extract all the files

First way:

gzip -d testdir.tar.gz => produces the decompressed testdir.tar

tar xf testdir.tar => produces the directory testdir

Second way:

tar xzvf testdir.tar.gz => produces the testdir

again, the - in front of xf or in front of xzvf is optional

note that we used 2 separate utilities: tar and gzip

to create testdir.tar.gz because we want to be able

to add more files to the tar file

and to "untar", we can use the same 2 utilites in reverse, or, we can use just one utility (tar) with the z option

====================

shell scripting

====================

create a variable called myvar that stores 10

myvar=10

==============================================

print myvar

echo $myvar

==============================================

change myvar to store the string hello there!

myvar='hello there!'

note that we need to use single quotes here

because the $ and ! are still interpreted by the

shell inside double quotes

==============================================

create a shell script that will do a long listing

of the current directory when you type list on the

command line

step 1. create a file called list (assume you don't have

a file called list already), with the following line

ls -l

step 2. give yourself execute permission

step 3. type list on the command line

=======================================

is there another way to run the list script?

bash list

======================================

what do you typically put as the first line of

the script?

#!/bin/bash

to indicate that a bash shell is needed to run

the script

======================================

is this required for the script that you just wrote?

why or why not?

no, ls -l is a standard linux command and is not

a bash shell programming construct, so it will

run with any shell, since any Unix/Linux shell

should understand ls -l

======================================

sanity check: you still want to type list to do

a long listing of the current directory. Is there

a simpler way other than writing a script?

yes, set up an alias

It will run faster because there is no need to fork

another bash process to run it

alias list='ls -l'

list # to run

or you can also store the command in a shell variable

and "run" the variable

list='ls -l'

$list

===================================================

write a shell script that prints the number of unique

users who are currently logged in

Then the script asks for a filename and appends to the

file the earliest login time

The script ends with a message that the file has

been created

who | sort | cut -d' ' -f1 | uniq | wc -l

echo -n 'filename: '

read $fName

who | sort +4 -5 | awk 'NR == 1 {print $5}' > $fName

====================================================

write a shell script to ask the user for a name (fist and last)

then print hello, followed by the first name and last name

when printing, add \* in front and in back of the name to make

the user feel 'special'

Add to the script so it prints out the user's login id and

default shell, with explanation

Add to the script so it prints the user's log in time

echo -n 'first name: '

read fName

echo -n 'last name: '

read lName

echo 'hello \*$fName $lName\*'

echo 'Your login id is: `whoami`'

echo 'Your default shell is `echo $SHELL`'

echo `who am i | awk '{print $5}'

sample midterm

========================================================

1. egrep (2 questions)

Print all lines that have the word linux, regardless of case, in fileA and fileB

egrep -i 'linux' fileA fileB

========================================================

Print the line numbers and lines that have no digits in fileA

Our first attempt:

egrep -n '^[^0-9]+$' fileA

Question in class: do we have to use anchors?

egrep -n '[^0-9]+' fileA

Answer: yes, we need anchors.

Without anchors, a line "1234abc356" will be printed when it shouldn't be

because the "abc" satisfies the regex of 1 or more non-digits

Question in class: what if we use \* instead of + ?

egrep -n '^[^0-9]\*$' fileA

Answer: it means blank (empty) lines will also be printed along

with lines that have no digits

Important question: can we make the first solution shorter?

egrep -nv '[0-9]' fileA

========================================================

2. find (2 questions)

Print a count of how many regular files you have in all your directories

find $HOME -type f | wc -l

Note: can use ~ instead of $HOME

=========================================================

Start from the labs directory, remove all links that point to dirA

Using -type l:

find labs -type l

this catches all links, but we only want links that point to dirA

A better choice is to use -lname, which looks for the path inside the

link to see if it ends with dirA. Both of these solutions are fine:

find labs -lname '\*dirA' | xargs rm

find labs -lname '\*dirA' -exec rm {} \;

Question: is there a difference when using '\*/dirA' vs '\*dirA' ?

find labs -lname '\*/dirA'

Answer: check the class exercise, we discussed this

Note that we can't use a pipe directly to rm.

rm is not a utility that accepts input from a pipe

find labs -lname '\*dirA' | rm

In general filters are good candidates for pipes

=========================================================

3. sed (3 questions)

fileA is a forwarded email that's been saved, where each line begins

with > and 1 space.

Print fileA without the "> " in front

Note: we don't need the address field because we want to process all lines

Our first attempt:

sed -n 's/^> (.\*)/\1/p' fileA

shorter:

sed -n 's/^> //p' fileA

shorter yet:

sed -n 's/> //p' fileA remove the ^ because we're told that every line

has "> " at the beginning

and the shortest:

sed 's/> //' fileA

Note that we can't use the command y:

sed -n 'y/> //

because every > and every space in the file will be changed

Note also that we can't use the command d:

sed 'd' fileA

because d will delete the entire line, we have to choice which part

of the line d will delete

=========================================================

Write a sed script that prints the first 4 lines of fileA, but with all

spaces changed into -

sed -f s2 fileA

s2:

1,4 y/ /-/

4q

=========================================================

4. awk (3 questions)

fileA has 10 fields, separated by semicolon (;)

Print fields 9, then 5, then 7 of all lines where field 5 > field 7.

When printing the field, separate them with colon (:)

Our first solution, using a script:

BEGIN { OFS = ":", FS = ";" }

$5 > $7 { print $9, $5, $7 }

Our second solution, using a command line:

awk -F ';' '$5 > $7 { print $9 ":" $5 ":" $7 }' fileA

========================================================

Write an awk script that prints the line number of all lines where the

fields add up to a negative number

script:

{

sum = 0 # don't forget to initialize sum for every line

# and it can't be initialized in the BEGIN block

for (i = 1; i <= NF; i++)

sum+= $i

if (sum < 0)

print NR

}

=========================================================

5. anything goes (4 questions)

Use the shortest command line to do the work (you will need

to decide which command will make it short)

print lines 3 to 8 of fileA, but with all white spaces removed from

the line

sed -nr '3,8 s/[[:space:]]//gp' fileA

Note the use of r option, and g and p flags for s

=========================================================

you have full permission for all your directories. For all your directories

where you already give execute permission for group and others,

add the read permission for group and others

2 solutions:

find ~ -type d -perm 711 -exec chmod 755 {} \;

find ~ -type d -perm 711 | xargs chmod go+r

========================================================

sample final

========================================================

1. It's the end of the quarter, you want an efficient way

to save all your files that you have on voyager so you can easily

recreate your entire voyager home directory on your linux system at home.

Show all the steps, starting from which directory you should be in

when you begin the save.

cd # go to home directory

tar czf homeDir.tgz \* # create tar file out of files in home dir

go to your linux system:

sftp your\_id@voyager.deanza.edu # use sftp to access voyager

<password on voyager>

get homeDir.tgz # fetch the file from voyager

exit

tar xzf homeDir.tgz # unarchive to get all files at your home directory

# on voyager and all their subdirectories

2. Given the following makefile.

PROGRAM = example.so

BINARY = $(PROGRAM)

EXAMPLEDLL = $(PROGRAM)

INCLUDEPATH = ../include

DLLDIR = .

RM = /usr/bin/rm

ECHO = echo

MODE = normal

SRCDIR = .

OBJS = xp\_echo.o

OBJECTDIR = .

CFLAGS = -I$(INCLUDEPATH)

$(EXAMPLEDLL) : $(OBJS)

@$(RM) $(DLLDIR)/$(EXAMPLEDLL)

@$(ECHO) "Loading $(EXAMPLEDLL)"

@$(ECHO) " MODE: $(MODE)"

@$(ECHO) " OBJS: $(OBJS)"

cd $(OBJECTDIR); \

$(CC) -o $(DLLDIR)/$(EXAMPLEDLL) $(OBJS)

@$(ECHO) "$(EXAMPLEDLL) done"

$(OBJS) : $(SRCDIR)/xp\_echo.c

$(CC) $(CFLAGS) -o $(OBJECTDIR)/$(OBJS) -c $(SRCDIR)/xp\_echo.c

a. Are there any phony targets?

no, there is no .PHONY

b. What actions take place when you type: make

What gets printed to screen?

- Look at the first rule to see that $(EXAMPLEDLL) depends on $(OBJS),

which is another target

- Find the ($OBJ) target and look at the 2nd rule to see that $(OBJS)

depends on $(SRCDIR)/xp\_echo.c

- Since $(SRCDIR)/xp\_echo.c is not a target, no need to follow it to get

to another rule

- If timestamp of $(SRCDIR)/xp\_echo.c is older than timestamp of

$(OBJS), then go to step A

- else run the command of the second rule to create a new $(OBJS)

- Step A

if timestamp of $(OBJ) is older than timestamp of $(EXAMPLEDLL)

then go to step B

- else run the commands of the first rule to create a new ($EXAMPLEDLL):

- rm example.so in the current directory

- print: Loading example.so

- print: MODE: normal

- print: OBJS: xp\_echo.o

- print: cd... line, then go to the current directory

- compile example.so

- print: example.so done

- Step B

done

3. Write a script that asks for a full path name of a directory

and prints how many levels down from root the directory is.

For example: /home/student/abc/dirA is 4 levels down from root

#!/bin/bash

echo Enter full path: # prompt the user

read path # read in absolute path

echo number of levels from root: # print explanation of output number

echo $path | awk -F'/' '{ if ($NF == "") {print NF - 2}

else {print NF - 1}}'

# send the path into awk so awk can separate each directory level into

# one field, and NF is the number of directories.

# the first field, $1, is always empty because the path starts with /

# so we subtract 1 from NF.

# but the last field can also be empty if the path ends with /

# so we use an if statement in awk to check if the last field is empty

# or not, and subtract 2 from NF if it is empty

=============================================

UTILITY OUTPUTS

=============================================

[ohericcc@voyager]$ ls -l

total 28

drwx---r-x 2 ohericcc unixStudent 4096 Mar 21 12:25 testdir

-rw-r--r-- 1 ohericcc unixStudent 3304 Mar 21 12:28 testdir.tar.gz

-rw----r-- 1 ohericcc unixStudent 5030 Mar 21 12:29 testfile

[ohericcc@voyager ~]$ ls -F

18b/ dist@ lab2.cleaned

[ohericcc@voyager mod6]$ who

sarah2016 pts/1 Mar 21 11:06 (108-202-244-91.lightspeed.sntcca.sbcglobal.net)

navneet07 pts/2 Mar 21 10:59 (10.41.83.253)

deepikam pts/3 Mar 21 09:12 (c-73-202-60-89.hsd1.ca.comcast.net)

[ohericcc@voyager mod6]$ whoami

ohericcc

[ohericcc@voyager ~]$ wc s1

13 21 149 s1 (lines, words, characters/bytes)

[wc -L gives longest line character count]

[ohericcc@voyager mod6]$ finger

Login Name Tty Idle Login Time Office Office Phone Host

20089862 Dunn Douglas James pts/18 3 Mar 21 12:04 (10.41.90.209)

ashleyk9318 Kim Jisoo pts/20 Mar 21 12:10 (10.41.81.50)

deepikam Metkar Deepika Sub pts/3 24 Mar 21 09:12 (c-73-202-60-89.hsd1.ca.comcast.net)

[ohericcc@voyager ~]$ tty (terminal id)

/dev/pts/7

===============

GOOD REFERENCES

===============

1. SYMBOLIC LINK

ln -s /home/distribution/cnguyen/cis18b link18