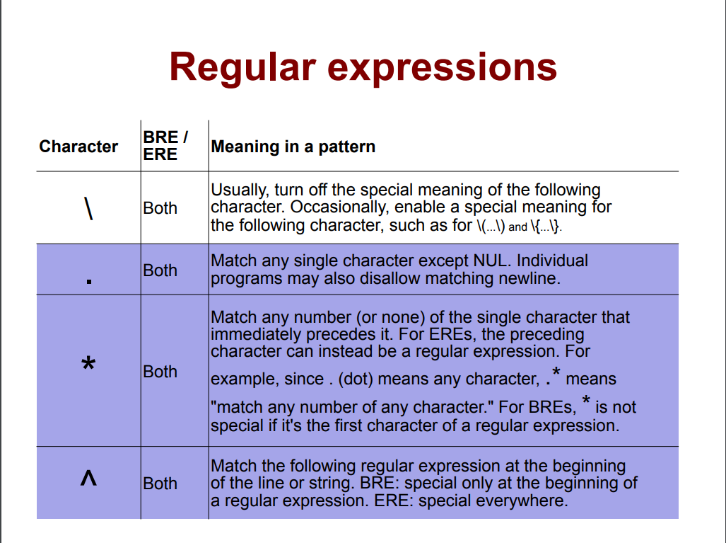
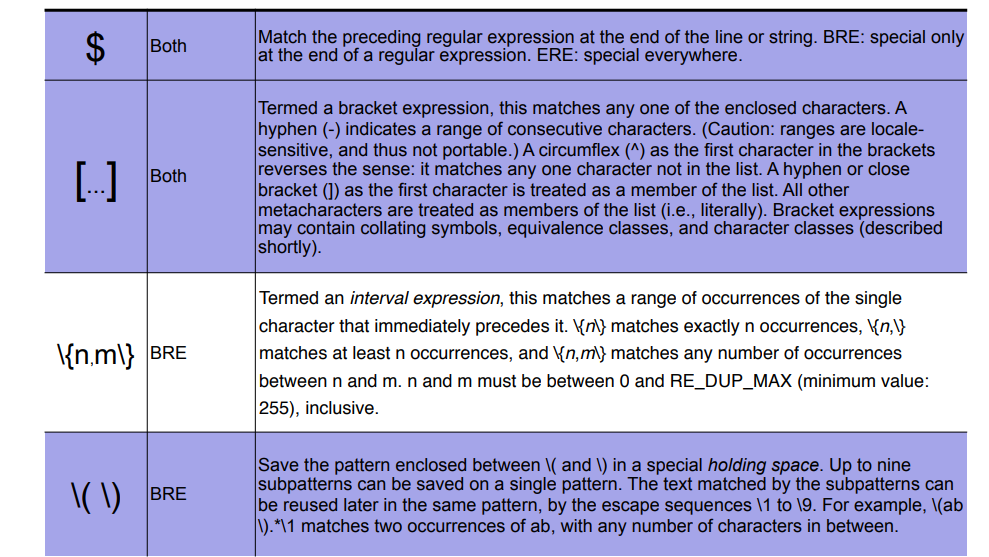
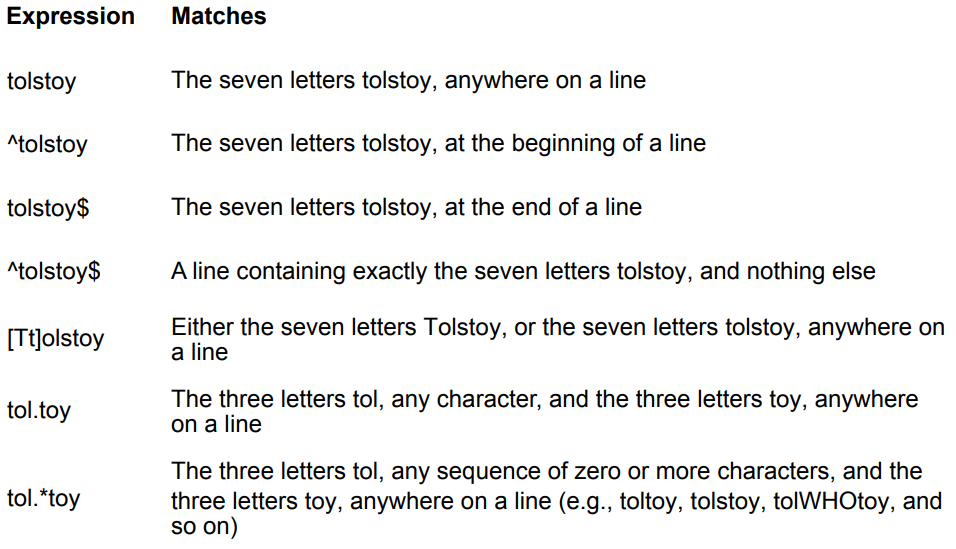
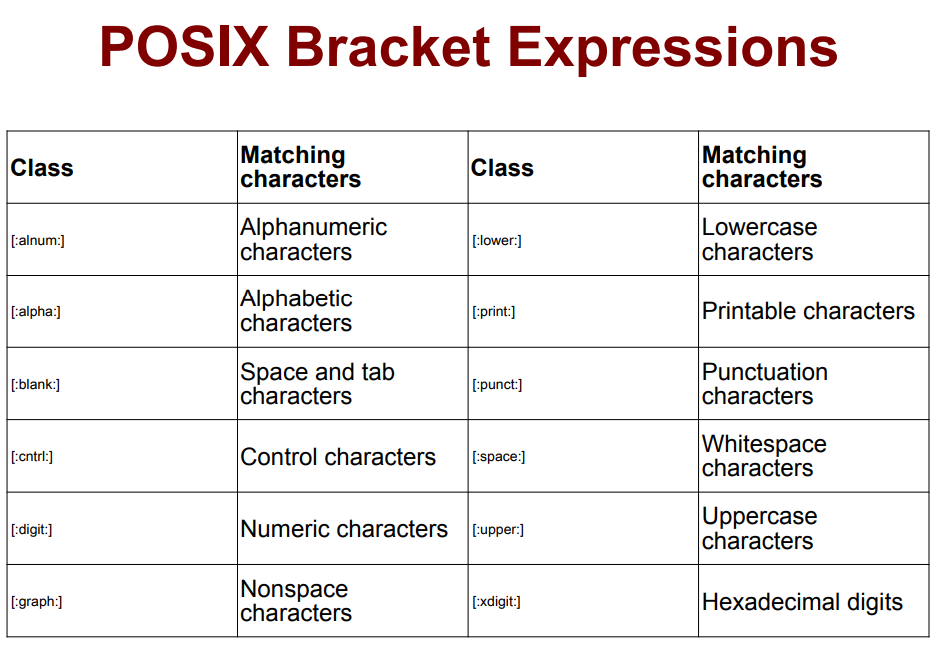
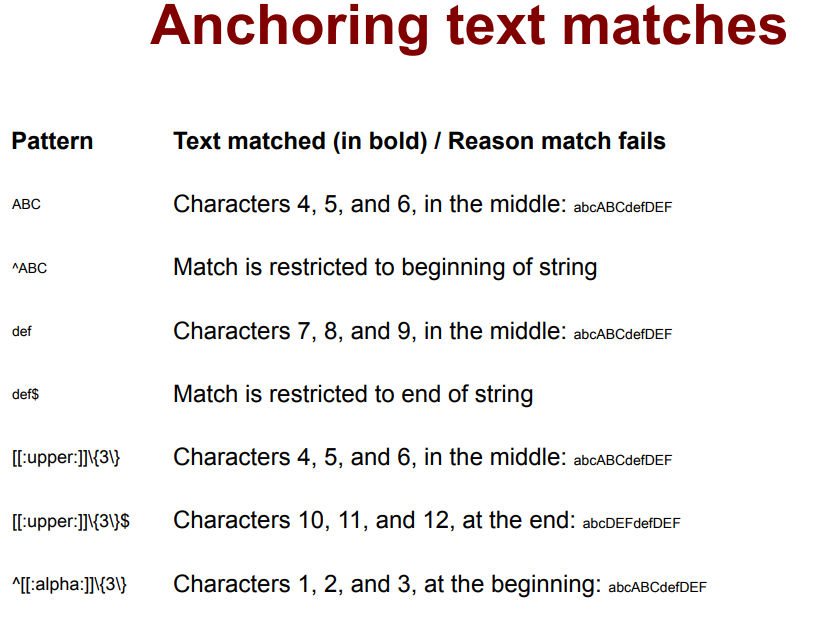
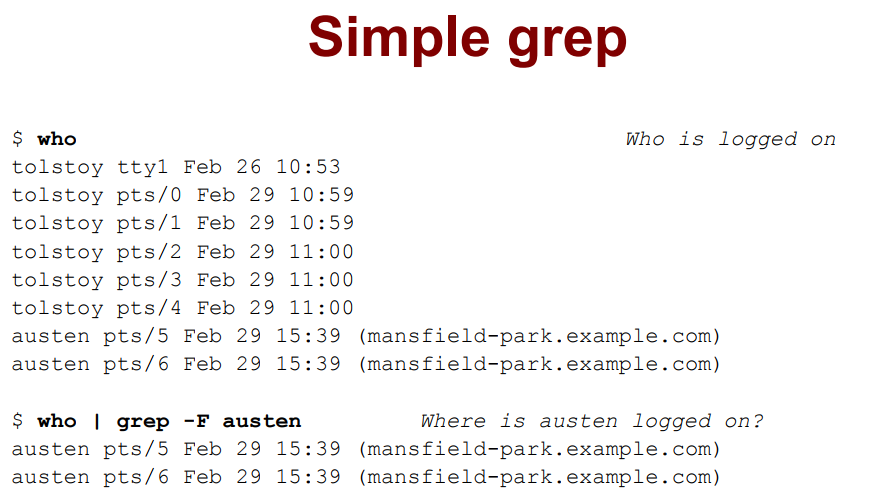
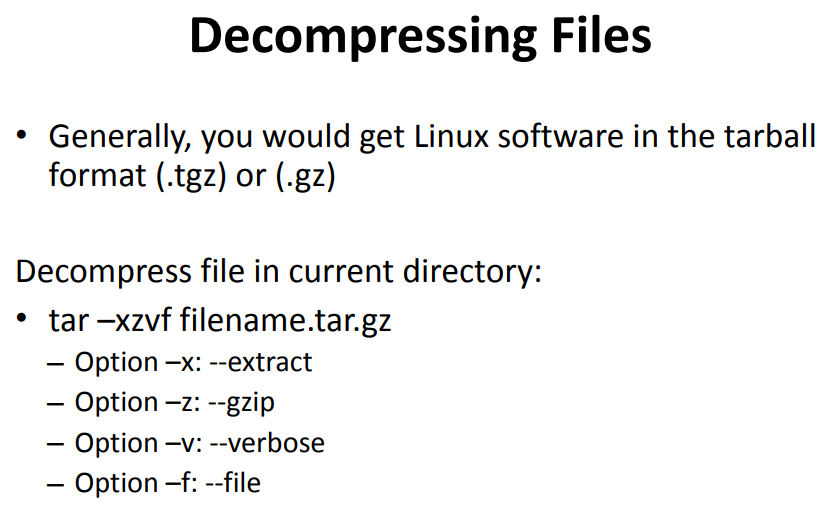
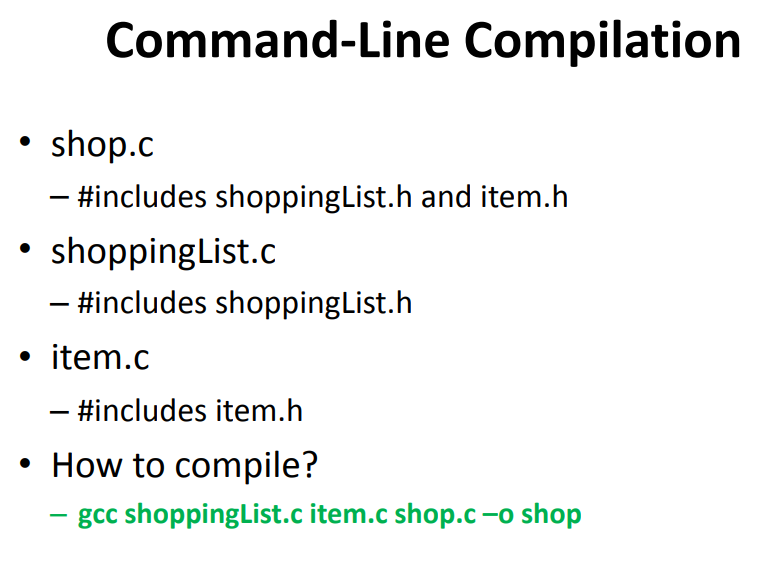
WEEK 2: BASH SCRIPTING + REGEX

|  |  |
| --- | --- |
| #!/bin/bash  #Delete everything from beginning of file to right before Adopt, first  #English word  sed '/<!DOCTYPE html/,/Adopt<\/td>/d' |  #Delete everything including and after the end of the table. The empty  #td tags to be deleted later  sed -e '/<\/table>/,$ d' |  #Delete u (underline) tags  #At first my implementation only deleted first instance, but StackOverflowed  #to make it a universal solution  sed 's/<u>//g' |  sed 's/<\/u>//g' |  #Replace ` with '  sed s/\`/\'/g |  #Uppercase to lowercase  tr '[:upper:]' '[:lower:]' |  #Delete English Words  sed '/<tr>/,/<\/td>/d' |  #Delete Leftover </tr> tags  sed '/<\/tr>/d' |  #Delete empty td tags at the end  sed '/<td><\/td>/d' | | ###############################################################  # At this point, we've done most of the filtering and just need to extract  # the Hawaiian words from the indented td tags.  ###############################################################  #Delete td tags  sed 's/<td>//' |  sed 's/<\/td>//' |  #Remove indents  sed 's/ //g' |  #Delete commas  sed 's/,//g' |  #Turn spaces into newlines  sed 's/ /\n/g' |  ##############################################################  # Done formatting, now to delete any incorrect words and sort.  ##############################################################  #Delete lines that contain invalid letters  sed '/[bcdfgjqrstvxyz]/d' |  #Sort, unique values displayed once each  sort -u |  #For whatever reason there's a blank line on top. Remove it  sed -e '1,1d' |

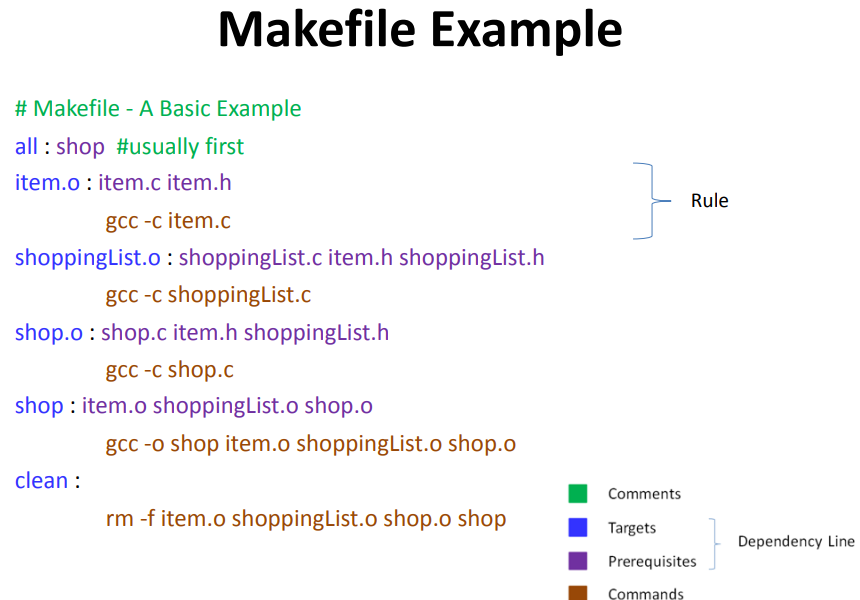
|  |  |
| --- | --- |
| #!/bin/bash  path=$1  #store list of files in RESULT  RESULT=`ls $path | sort` #all the normal files  A=`ls -a $path | grep '^\.' | sort` #all the files that start with .  #declare array ARRAY to specify which files have duplicates  declare -a ARRAY  let count=0  #recognize filenames containing spaces  IFS=$'\n'  #filter out the files that we don't want starting with .  for AF in $A  do  if [ ! -r "$path/$AF" ] #no read permission  then  echo "Unable to read $path/$AF"  elif [[ ! -L "$path/$AF" && -f "$path/$AF" ]] #regular & not symbolic  then  ARRAY[$count]="$path/$AF"  let count=count+1  fi  done | #same thing for rest of files  for F in $RESULT  do  if [ ! -r "$path/$F" ] #no read permission  then  echo "Unable to read $path/$F"  elif [[ ! -L "$path/$F" && -f "$path/$F" ]] #regular & not symbolic  then  ARRAY[$count]="$path/$F"  let count=count+1  fi  done  for (( k=0; k<$count; k++ ))  do  for (( j=$k+1; j<$count; j++ ))  do  cmp -s "${ARRAY[$k]}" "${ARRAY[$j]}"  if [ $? -eq 0 ]  then  ln -f -- "${ARRAY[$k]}" "${ARRAY[$j]}" #link 2nd to 1st  ARRAY[$j]=ARRAY[$k] #so that later iterations get it right  fi  done  done |





WEEK 3: MAKEFILES & PYTHON

Make

• Utility for managing large software projects

• Compiles files and keeps them up-to-date

• Efficient Compilation (only files that need to be recompiled)

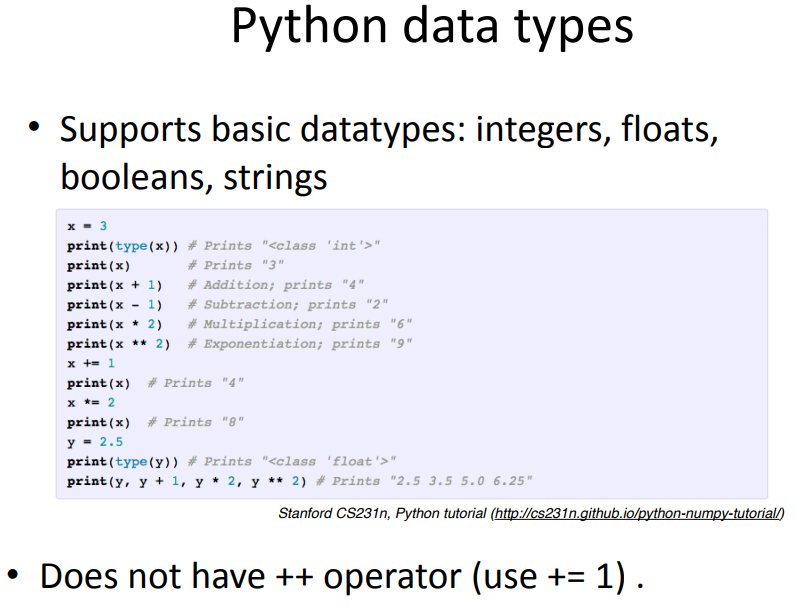
Build Process

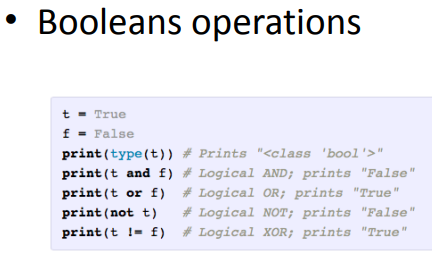
• configure – Script that checks details about the machine before installation

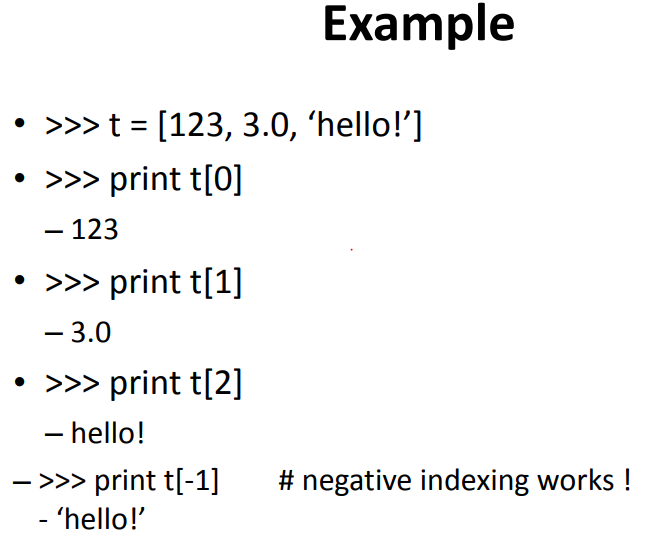
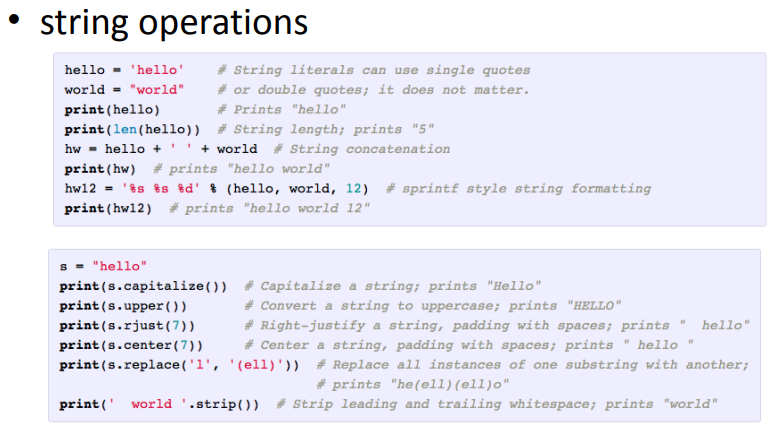
• Dependency between packages – Creates ‘Makefile’

• make – Requires ‘Makefile’ to run – Compiles all the program code and creates executables in current temporary directory

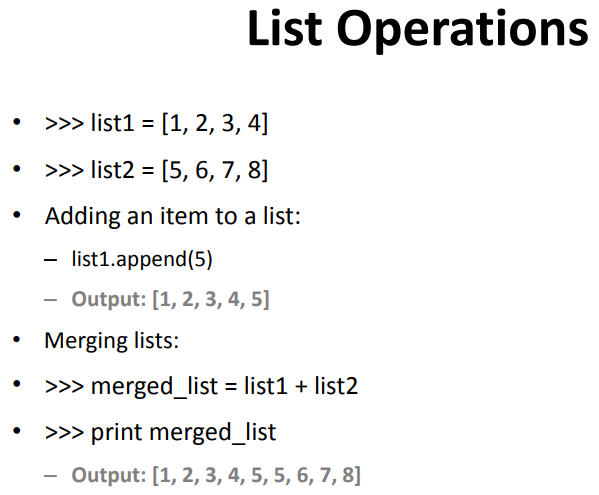
• make install – make utility searches for a label named install within the Makefile, and executes only that section of it – executables are copied into the final directories (system directories)

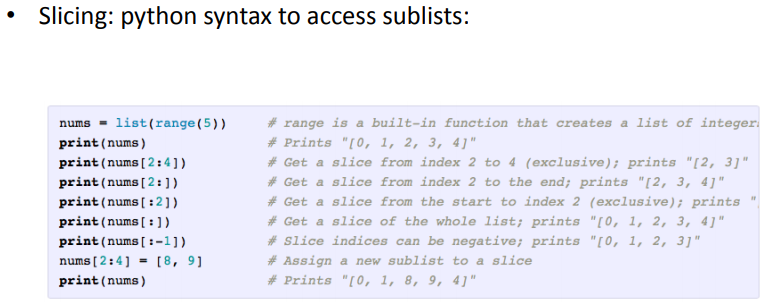


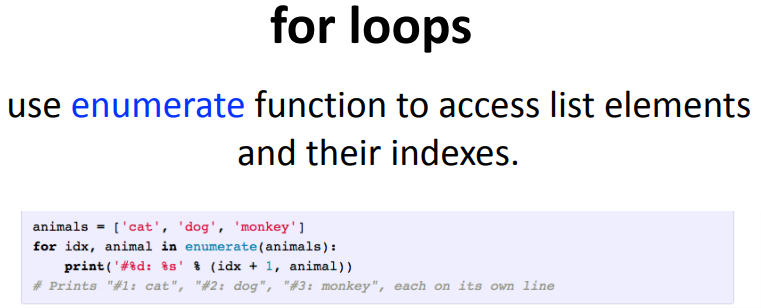
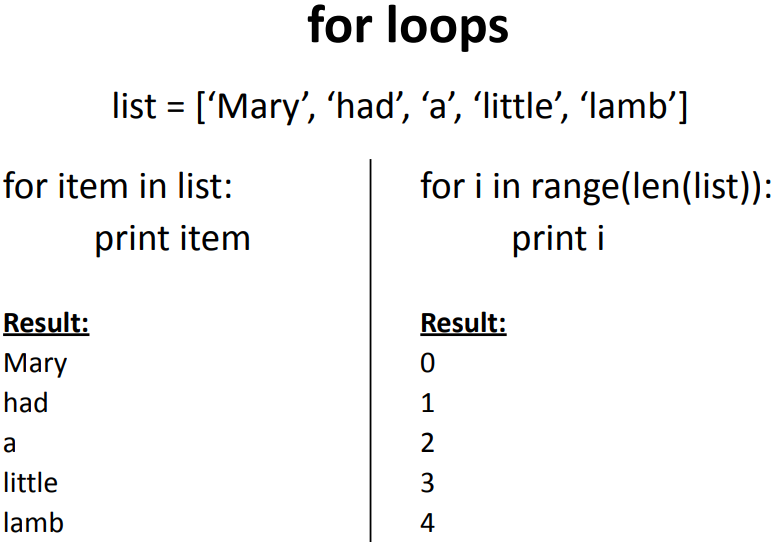


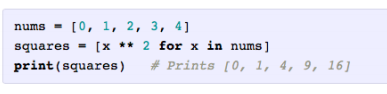


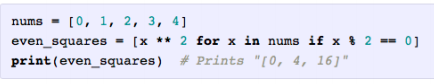
• List: python equivalent of an array, but it is resizable and can contain elements of different types.

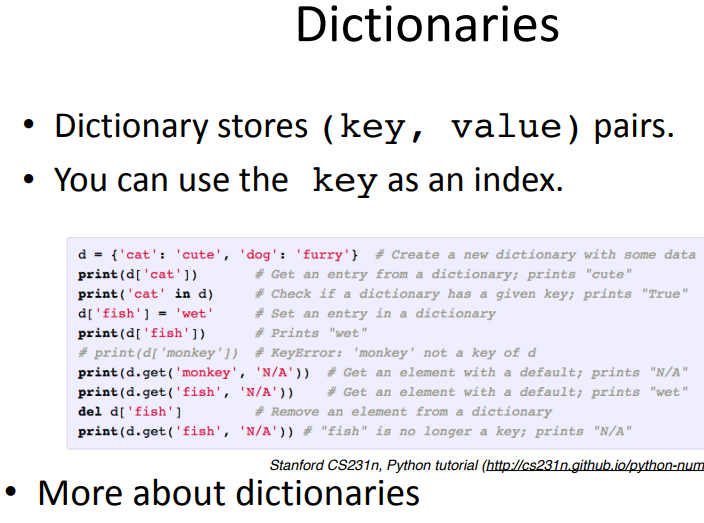
How to access elements? – List\_name[index]

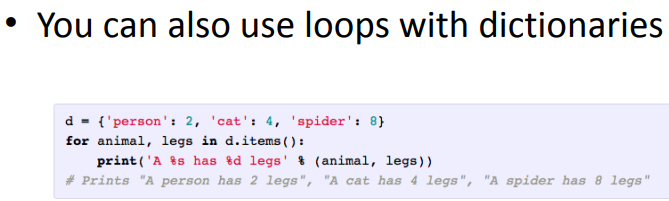








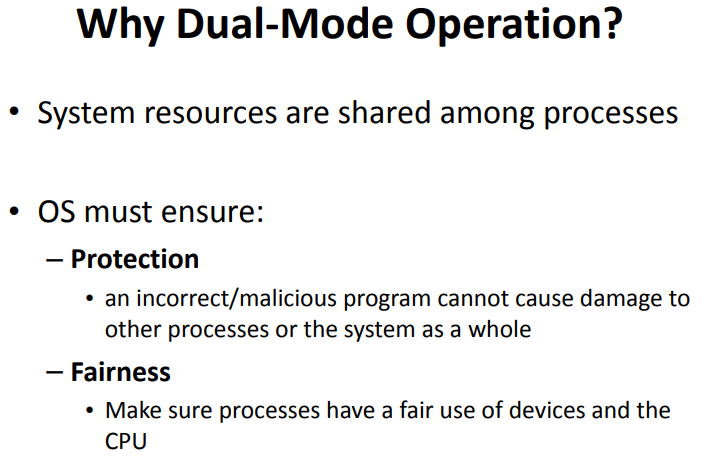


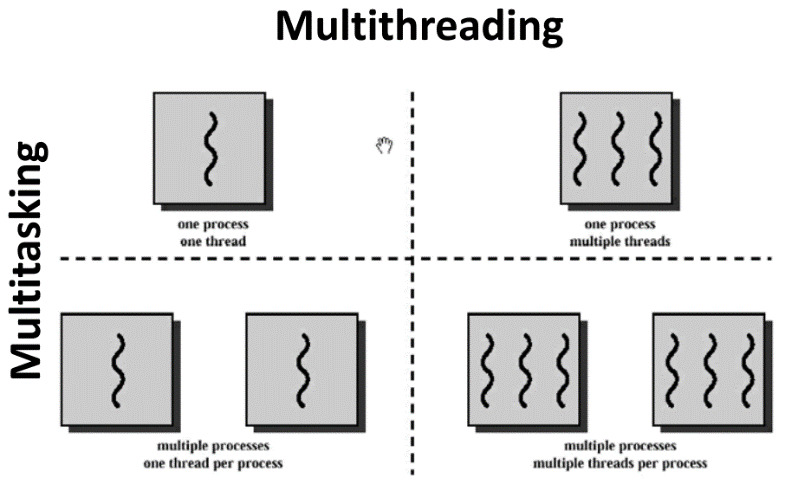


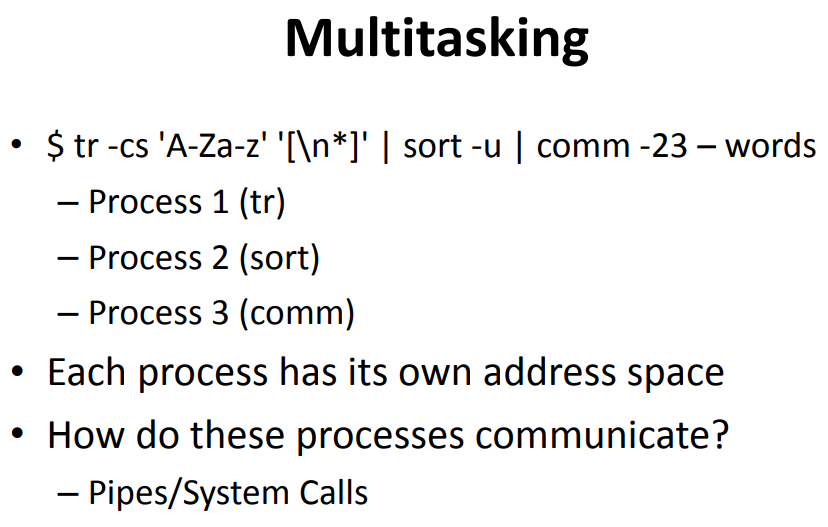
WEEK 4: C PROGRAMMING & DEBUGGING

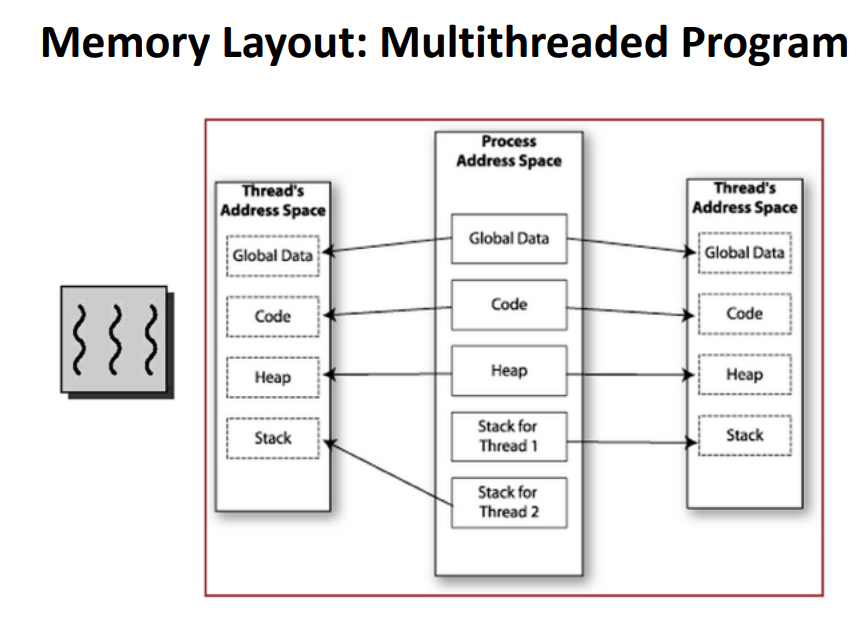
See Jahan’s notes

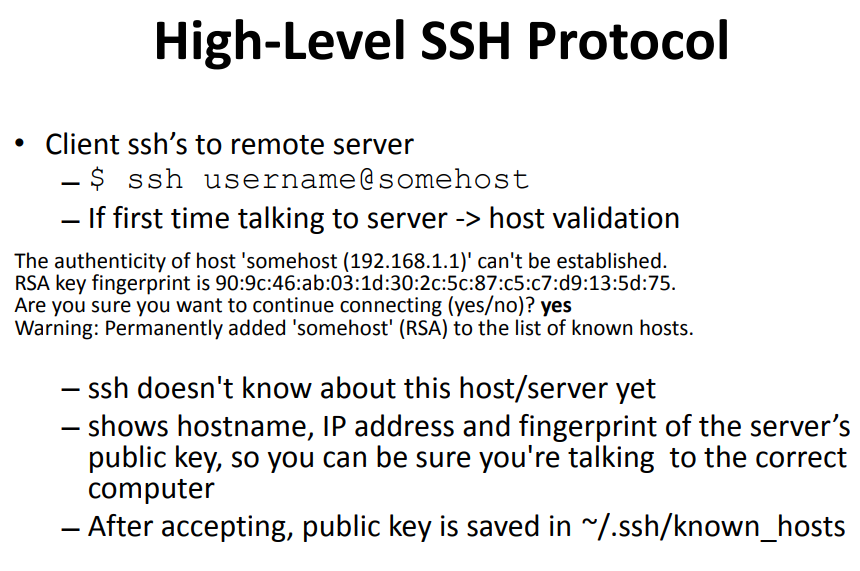
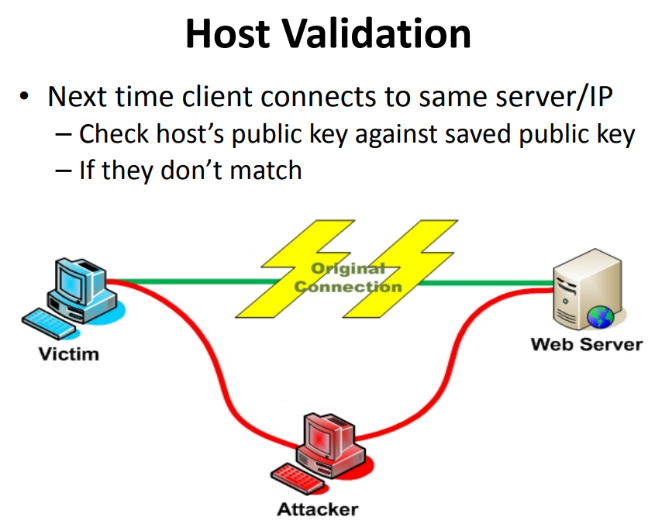
WEEK 5: SYSTEM CALL PROGRAMMING & DEBUGGING

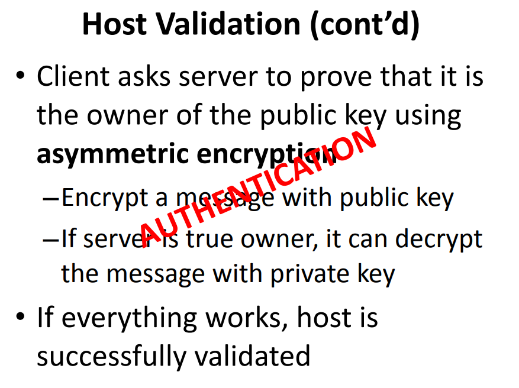
See Jahan’s notes for details

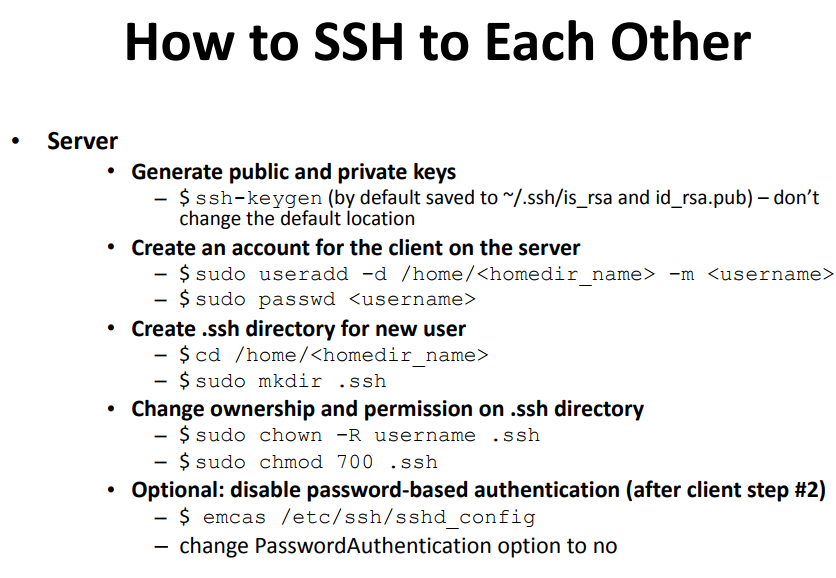
WEEK 6: MULTITHREADED PERFORMANCE

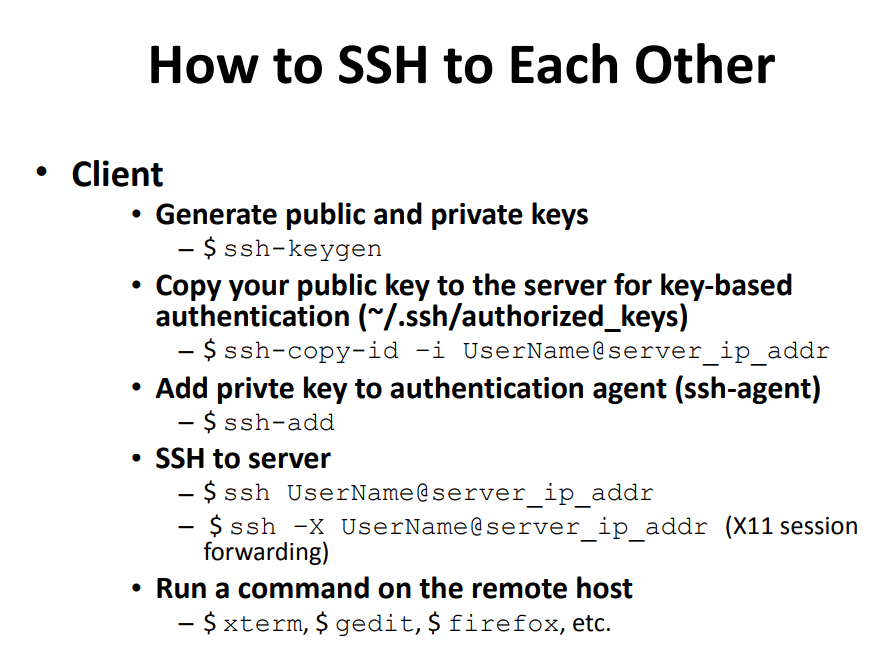


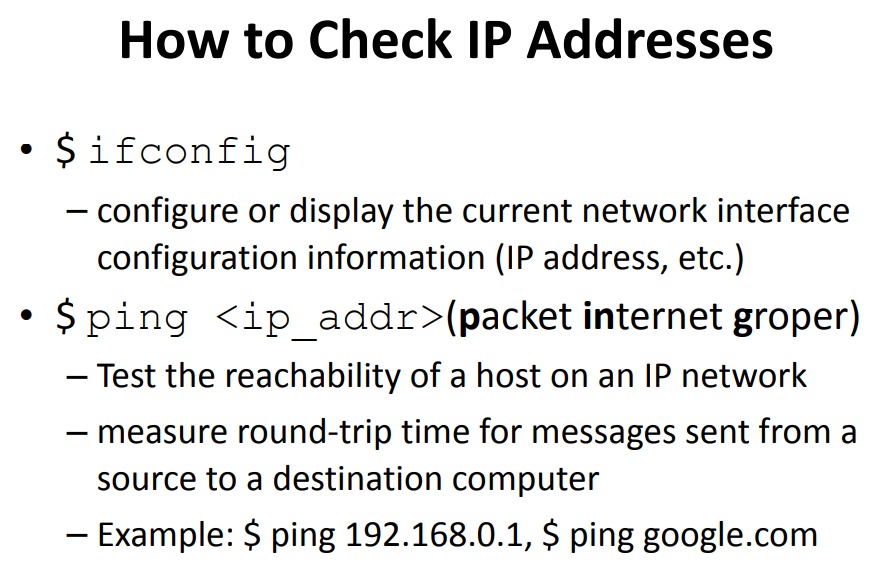


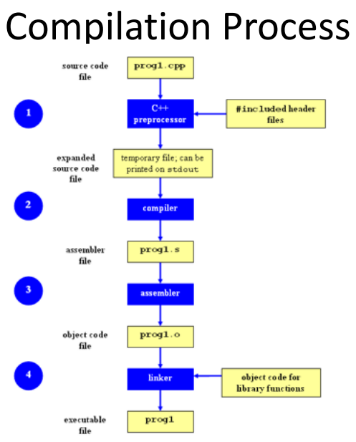
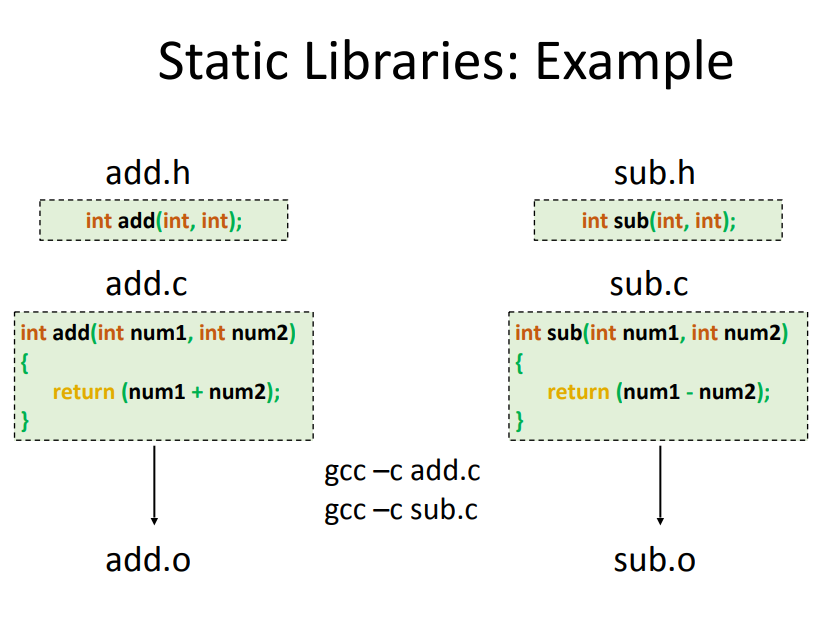
WEEK 7: SSH

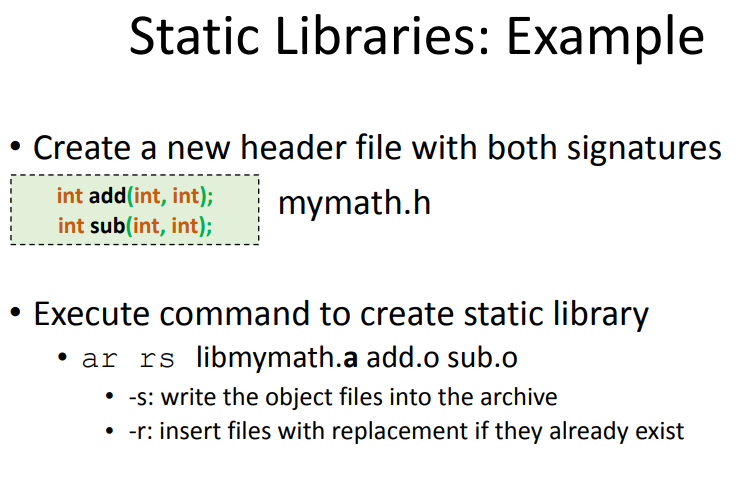


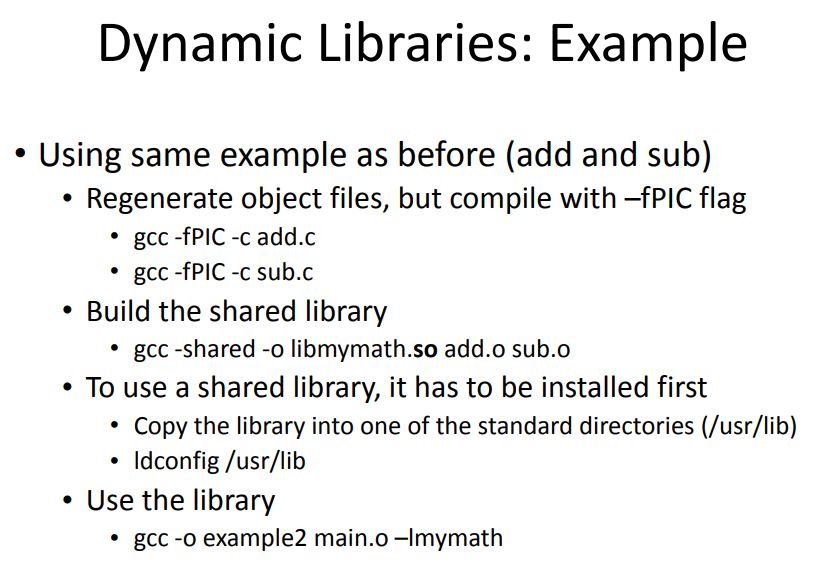
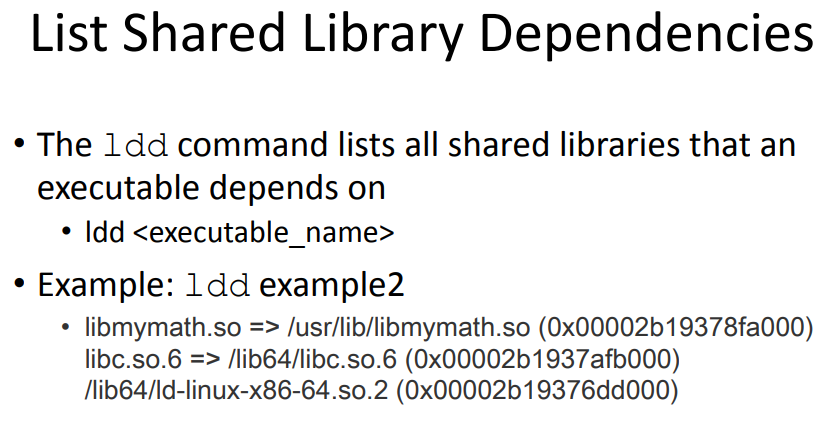


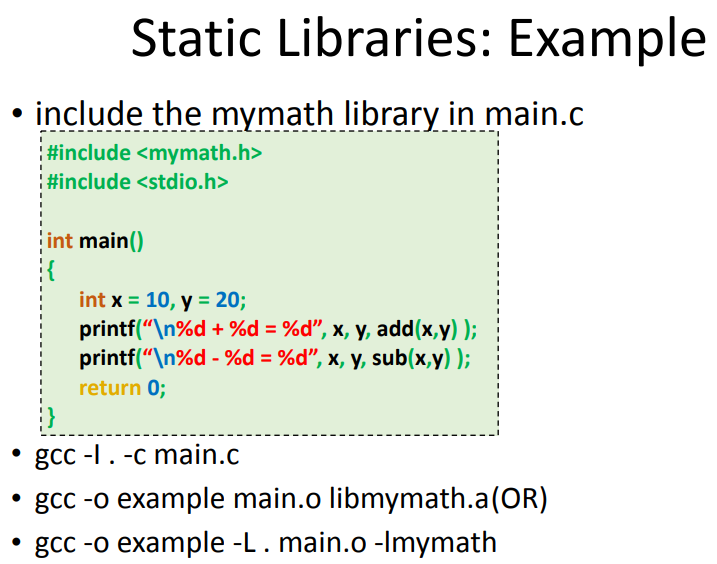


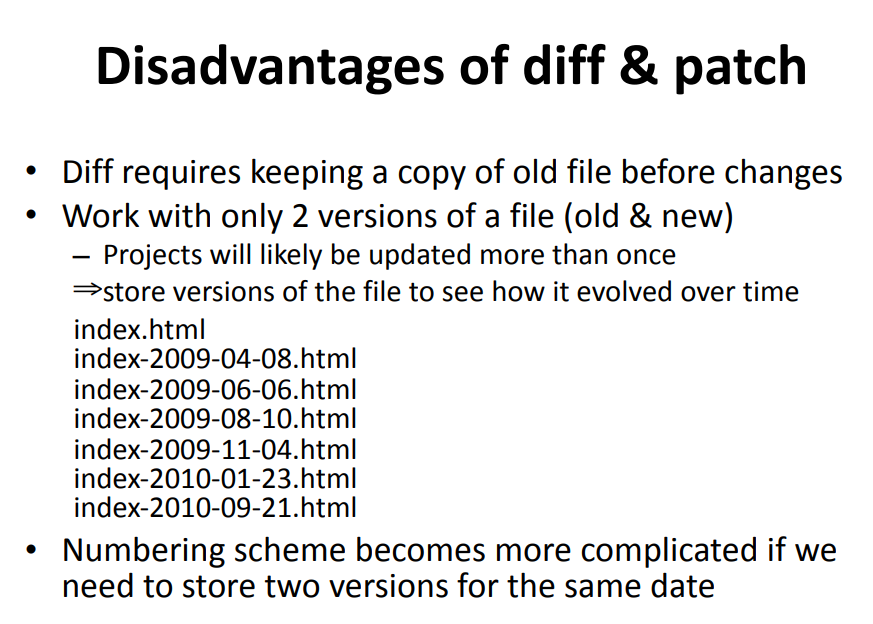


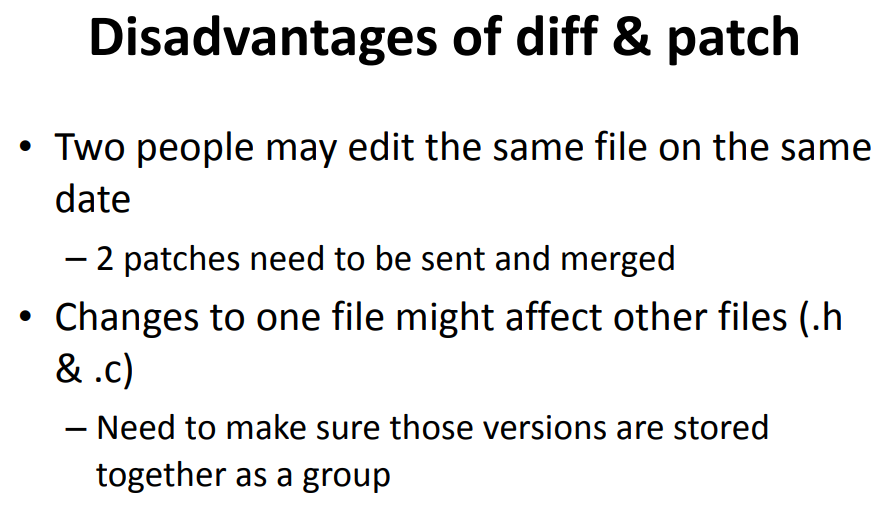
WEEK 8: DYNAMIC LINKING

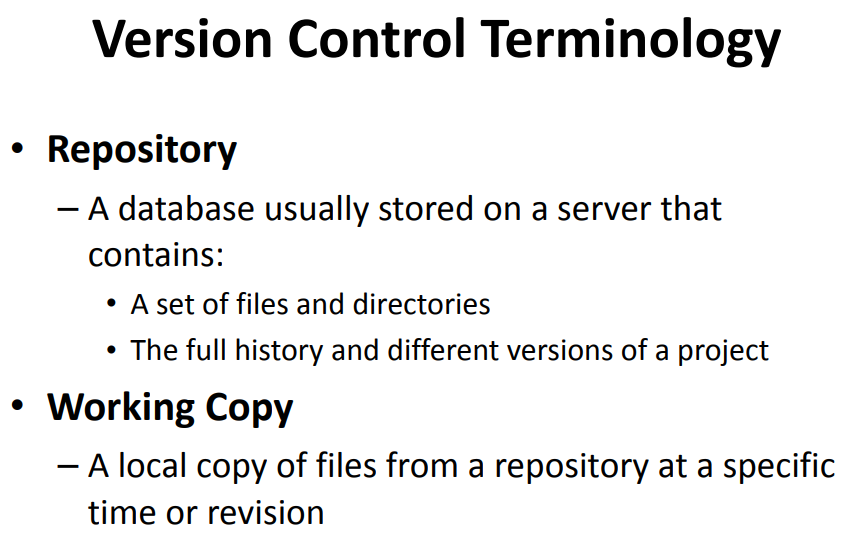


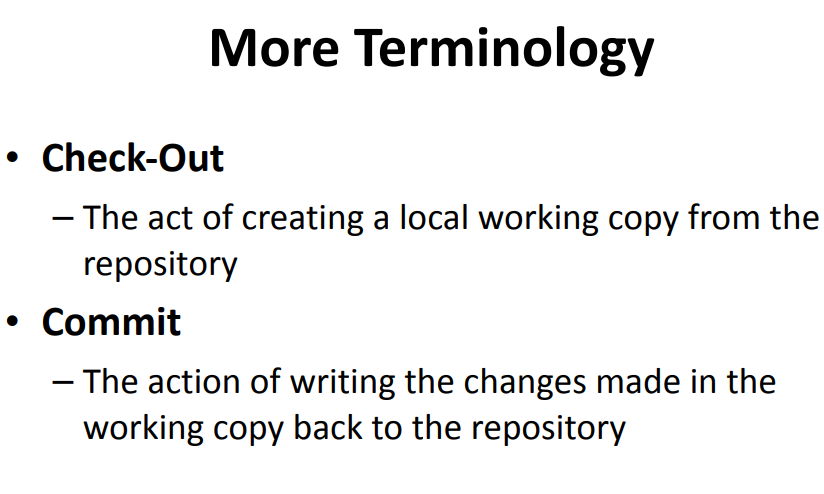


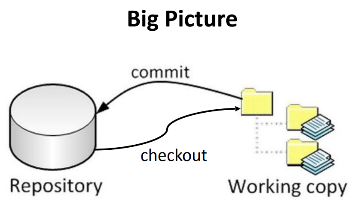
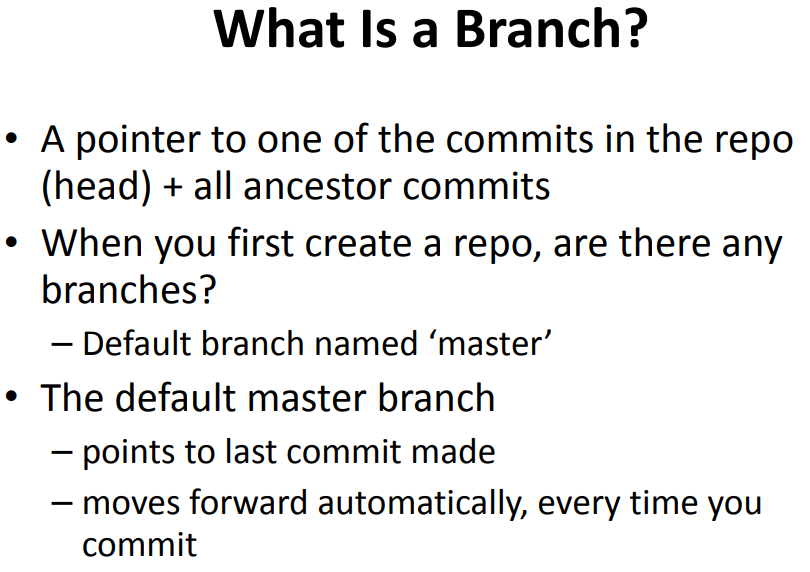
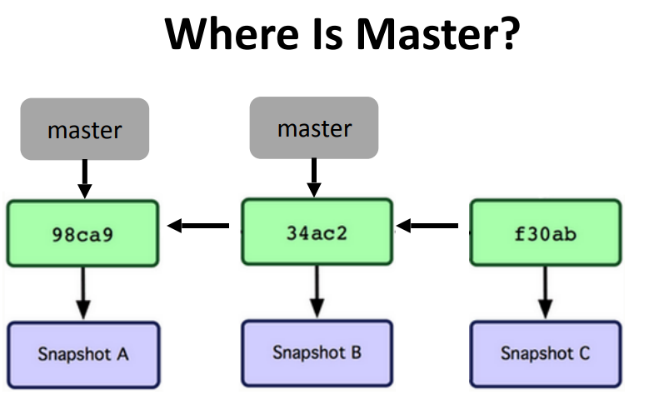


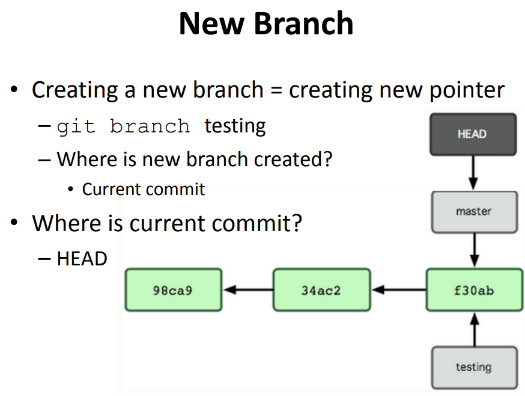
WEEK 8: DYNAMIC LINKING











**Why Branching?**

• Experiment with code without affecting main branch

• Separate projects that once had a common code base

• 2 versions of the project