WEEK 1

Open Source Software: publicly available source code that anyone can modify

GNU/LINUX (open-source OS like Ubuntu): Kernel, Shell, Programs

- **Kernel**: core of OS, allocates time/memory to programs, handles file system + communication between software and hardware, has most privileges
- **Shell**: interface between user and kernel, interprets + carries out user commands
- **Programs**: how to talk to kernel
- Why: fair allocation of resources to multiple users, prevents user access to kernel, takes care of some of the work

GUI vs CLI (graphic user interface vs. command-line interface (to access OS services))

- **GUI**: Intuitive, limited control, easy multitasking, bulky remote access
- **CLI**: Steep learning curve, pure control via scripting, hard multitask, easy remote

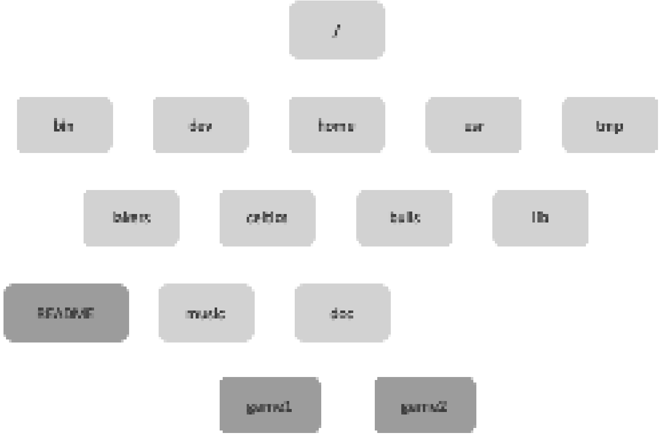
Files and Processes (everything is either a process or a file)

- **Process**: an executing program identified by PID
- **File:** collection of data (e.g. document, text, executable, directory, devices)

Absolute vs. Relative Paths: (from root / vs. from current directory)

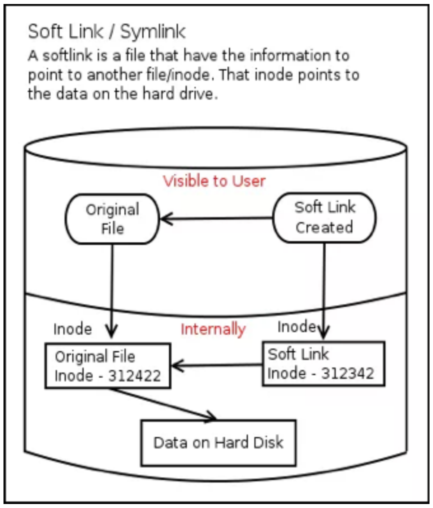
Important Directories:

/	root	/lib	essential libraries
/bin	binaries, fundamental utilities	/opt	locally installed software
/boot	files needed for successful booting process	/usr	user file system
/dev	devices	/var	variable
/etc	system-wide config files and system databases	/log	system log files
/home	user home directories		



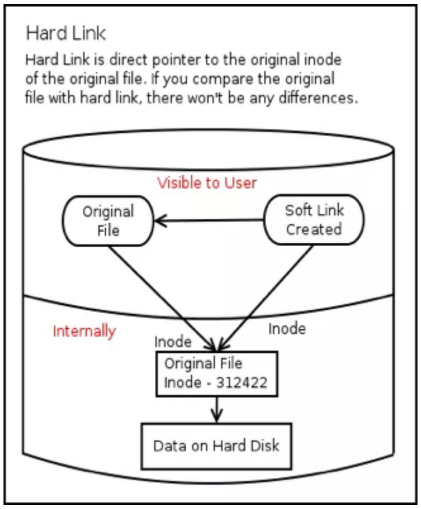
Changing File Attributes/Basic Commands

- **ln**: create a link
  - Hard link: point to physical data, have same inode #, only for files
    - inode is data structure with file info, ls -i file_name
  - Soft link (-s): symbolic link, points to a file/directory
    - Broken when original file/target DNE
    - Target deleted/renamed
    - Path element deleted/renamed/moved



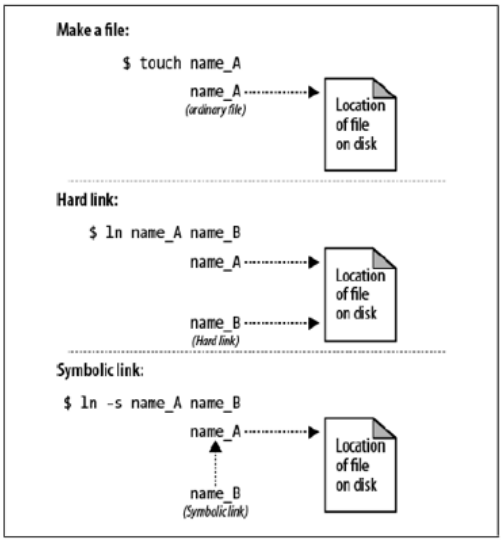
Soft Link / Symlink

A softlink is a file that have the information to point to another file/inode. That inode points to the data on the hard drive.



Hard Link

Hard Link is direct pointer to the original inode of the original file. If you compare the original file with hard link, there won't be any differences.



Make a file:

```
$ touch name_A
```

Hard link:

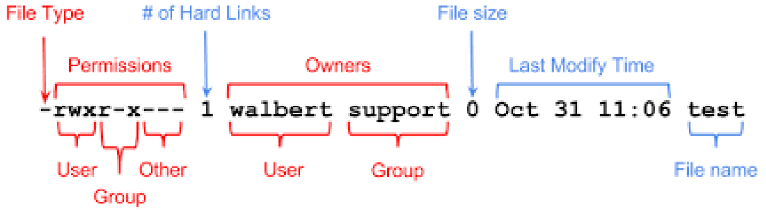
```
$ ln name_A name_B
```

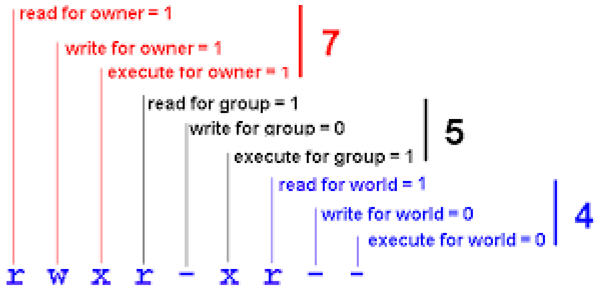
Symbolic link:

```
$ ln -s name_A name_B
```

- **touch**: update access and modification time to current time
  - touch file_name
  - touch -t 201101311759.30 file_name
    - Change access/modification time to 2011, Jan 31 at 17:59:30
- **find**
  - Options
    - -type: type of file (-f file, -d directory, -l link) (-executable)
    - -perm: permission of a file (-perm /u=r, -perm /a=x, -perm 777)
    - -name: name of a file
    - -prune: don’t descend into a directory
    - -o: or
    - -user: owner of a file (-user bob)
    - -maxdepth: how many levels to search
    - -mtime #
      - -#: looks for files modified in last # days
      - +#: looks for files modified before last # days
    - -atime #
      - -#: accessed in at least the past # days
      - +#: accessed in at most the past # days
  - File Name Matching
    - ?: match any single character in a filename
    - *: match 1+ characters in a filename
    - []: match any one of the chars between the brackets, use ‘-’ to separate range of consecutive chars

    - Examples
        - find . -name my*
        - find . -name my* -type f
        - find / -type f -name myfile -print
- **man <command_name>**
    - Documentation preinstalled with all Unix OSs, info about a linux command
    - man <section> <command_name> (1 user commands 2 System calls 3 C library etc)
    - man -k <word>: print all commands with specific <word> in man pages
- **File Permissions**
    - **Owner:** Determine what actions the owner of the file can perform on the file
    - **Group:** Determine what actions a user in the group can perform on the file
    - **Others:** Determine what actions all other users can perform on the file
    - setuid, setgid (SUID/SGID, u+s, g+s)
        - Appears as the letter ‘s’ in the permissions bit where owners execute permission is (-r-sr-xr-x)
        - Capital S in execute position instead of lowercase s indicates that execute bit is NOT set
        - Execute as owner/group permissions, not the user
    - sticky bit (o+t)
        - On shared directories, locks files within the directory from being modified by users other than the file creator, directory owner, or root
    - **chmod (symbolic)**
        - read (r), write (w), executable (x)
        - User, group, others
    - **chmod (numeric)**
        - example: chmod ug+rw mydir, chmod a-w myfile
        - example: chmod ug=rx mydir, chmod 664 myfile





Reference	Class	Description
u	user	the owner of the file
g	group	users who are members of the file's group
o	others	users who are not the owner of the file or members of the group
a	all	all three of the above, is the same as *ugo*

Operator	Description
+	adds the specified modes to the specified classes
-	removes the specified modes from the specified classes
=	the modes specified are to be made the exact modes for the specified classes

Mode	Name	Description
r	read	read a file or list a directory's contents
w	write	write to a file or directory
x	execute	execute a file or recurse a directory tree

#	Permission
7	full
6	read and write
5	read and execute
4	read only
3	write and execute
2	write only
1	execute only
0	none

WEEK 2

locale: Set of parameters that define a user’s cultural preferences (language, numbers)

- Prints info about current locale environment to standard output
- Gets info from LC_CTYPE, LC_COLLATE, LC_TIME, LC_NUMERIC, LC_MONETARY, LC_MESSAGES

Environment Variables: variables that can be accessed from any child process

- HOME: path to user’s home directory
- PATH: list of directories to search in for command to execute
- LC_TIME: date and time formats
- LC_NUMERIC: non-monetary numeric formats
- LC_COLLATE: order for comparing and sorting
    - ‘C’: ASCII order
    - ‘en_US’: case insensitive, else if same then uppercase is earlier
- change value: export VARIABLE=...

Text Processing Commands (sort, comm, tr)

- **sort**: sorts **lines** of text files depending on locale
    - -u: unique results
- **comm**: compare two **sorted** files **line by line** depending on locale
    - -: takes in from stdin
- **tr**: translate **or** delete chars
    - -c: complement, tr works on complement of specified chars (NOT)
    - -s: replaces sequence of repeated chars in set1 with single instance of char in set2

Compiled vs. Interpreted Language

- Compiled (e.g. C/C++)
    - source to machine code, executed by hardware
    - fast, efficient
    - requires recompilation
- Interpreted (e.g. PHP, Ruby, Bash)
    - shell reads out commands and actions as it goes
    - slower in execution
    - portable

Shell Scripting (bash, sh, csh, ksh)

- Shell is user interface to OS, commands taken as text and interpreted
- Shell script file is a file with shell commands, when executed new child “shell” process spawned to run it
    - First line states which child “shell” to use
    - #! /bin/sh
    - #! /bin/bash
- Enable execution with chmod +x script.sh
    - Otherwise: Permission Denied Error
- Execution Tracing (print out each command as it is executed)
    - set -x: turn tracing on
    - set +x: turn tracing off
- IF statements
    - Use test command or [] for expressions
    - if [test]; then … else … fi
- LOOPS
    - While loop

        - while [$COUNT -gt 0]; do … done
    - For loop
        - for f in $temp; do … done
        - for ((i=0; i < $count; i++)); do … done
    - let count=count+1
- Quotes
    - Single ‘’: literal meaning, no expansion
    - Double “”: expand backticks and $
    - Backticks `` or $(): expand as shell commands

Standard Streams

- Each program has 3 streams to interact with world
    - stdin (0): data going into program
    - stdout (1): where program writes its output data
    - stderr (2): where program writes its error messages
- Redirection/Pipelines
    - program < file: redirects file to program's stdin
    - program > file: redirects program’s stdout to file
    - program 2> file: redirects program's stderr to file2
    - program >> file: appends program’s stdout to file
    - program1 | program2: assigns stdout of program1 as stdin of program2

REGEX: notation that lets you search for text with a particular pattern

- Quantification
    - How many times of previous expression
        - ?: 0 or 1
        - *: 0+
        - +: 1+
- Grouping
    - Which subset of previous expression
        - Grouping operator ()
- Alternation
    - Which choices
        - Operators [] and |
- Anchors
    - Where
        - ^: beginning
        - $: end
- TABLES

grep: matches **lines**, print lines matching pattern (regex)

- uses basic regular expressions (BRE)
    - meta-characters ?, +, {, |, (,) lose special meaning, instead use backslashed versions
- egrep (grep -E): uses extended regular expressions (ERE), no backslashes needed
- fgrep (grep -F): matches fixed strings instead of regular expressions

sed: replace parts of text, filter and transform

- sed ‘s/**regex**/**replacetext**/[g]’

Text Processing Tools

- wc, head, tail, tr, grep, sort, sed

Location of pattern:

- ^ - beginning of line
- $ - end of line

Quantification (how many times do I want to match a pattern):

- . - Exactly 1 character (match one single character)
- * - Match 0 or more instances of the pattern
- + - 1 or more instances of pattern
- ? - 0 or 1 instances of pattern
- {n} - Match exactly n occurrences of pattern
- {n,m} - Match n to m instances of pattern (inclusive)
- {n,} - At least n occurrences of pattern

Alternation (Express different options of patterns to match):

- [ ] - Whatever is inside here, match only one character!
    - EX: [abc] - a or b or c
- | - OR
    - EX: (b+|c+) = 1 or more b OR 1 or more c. It allows you to match words

Grouping:

- () - Group a part of a pattern
    - EX: ab+ = a and 1 or more b
        - (ab)+ = 1 or more ab

There are two types of Regex's:

Basic Regex - You have to escape some of the Regex symbols to give them the special meaning in REGEX.

- Symbols to worry about: ?,+,|,{,},(,) - escape to use as REGEX

Extended Regex - Opposite, if you want to use the literal, escape, but otherwise you're defaulted to REGEX.

- Symbols to worry about: ?,+,|,{,},(,) - escape to use as literals

grep - Default as basic, egrep is extended and fgrep is fixed (doesn't understand REGEX)

- Example: Say we have a bash script with comments (has # symbols) and we want to find the comments TODO and FIXME.
  ```
  cat script | egrep ‘^#(TODO|FIXME)’
  cat script | grep ‘^#\(TODO\|FIXME\)’
  cat script | fgrep ‘#TODO’
               fgrep ‘#FIXME’
  ```

sed - find pattern and replace it!

- sed 's/*Regex to find things to replace*/*replace with*/'
- (the s means substitute, also note that the delimiter can be replaced with any symbols you i.e you don't have to use slashes!)
- Example: Find comments from bash script and remove them, but don't remove the shabang line!
    - `sed 's/^#[^!].*//'` = find # at start, not ! and any character
- Example: From a file of phone numbers, replace digit only form. Find (xxx) xxx-xxxx and make xxxxxxxxxx. Note the group () stores what’s inside it for back referencing, but remember sed is basic regex so you have to escape things
    - `sed 's/(\([0-9]\{3\}\))\([0-9]\)\{3\}-\([0-9]\{4\}\)/\1\2\3/' < contacts.txt`
- Note that sed does in place replacement!

POSIX Built-in Shell Variables

Variable	Meaning
#	Number of arguments given to current process.
@	Command-line arguments to current process. Inside double quotes, expands to individual arguments.
*	Command-line arguments to current process. Inside double quotes, expands to a single argument.
- (hyphen)	Options given to shell on invocation.
?	Exit status of previous command.
$	Process ID of shell process.
0 (zero)	The name of the shell program.
!	Process ID of last background command. Use this to save process ID numbers for later use with the *wait* command.
ENV	Used only by interactive shells upon invocation; the value of $ENV is parameter-expanded. The result should be a full pathname for a file to be read and executed at startup. This is an XSI requirement.
HOME	Home (login) directory.
IFS	Internal field separator; i.e., the list of characters that act as word separators. Normally set to space, tab, and newline.
LANG	Default name of current locale; overridden by the other LC_* variables.
LC_ALL	Name of current locale; overrides LANG and the other LC_* variables.
LC_COLLATE	Name of current locale for character collation (sorting) purposes.
LC_CTYPE	Name of current locale for character class determination during pattern matching.
LC_MESSAGES	Name of current language for output messages.
LINENO	Line number in script or function of the line that just ran.
NLSPATH	The location of message catalogs for messages in the language given by $LC_MESSAGES (XSI).
PATH	Search path for commands.
PPID	Process ID of parent process.
PS1	Primary command prompt string. Default is "$ ".
PS2	Prompt string for line continuations. Default is "> ".
PS4	Prompt string for execution tracing with set -x. Default is "+ ".
PWD	Current working directory.

Exit: Return value

Check exit status of last command that ran with $?

Value	Typical/Conventional Meaning
0	Command exited successfully.
> 0	Failure to execute command.
1-125	Command exited unsuccessfully. The meanings of particular exit values are defined by each individual command.
126	Command found, but file was not executable.
127	Command not found.
> 128	Command died due to receiving a signal

```
#!/bin/bash
if [ 5 –gt 1 ]
then
                echo "5 greater than
1"
else
                echo "not possible"
fi
```

- **While loop**

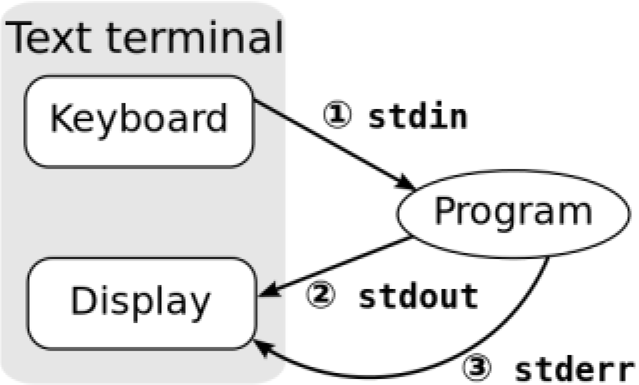
```
#!/bin/sh
COUNT=6
while [ $COUNT -gt 0 ]
do
                echo "Value of count is:
$COUNT"
                let COUNT=COUNT-1
done
```

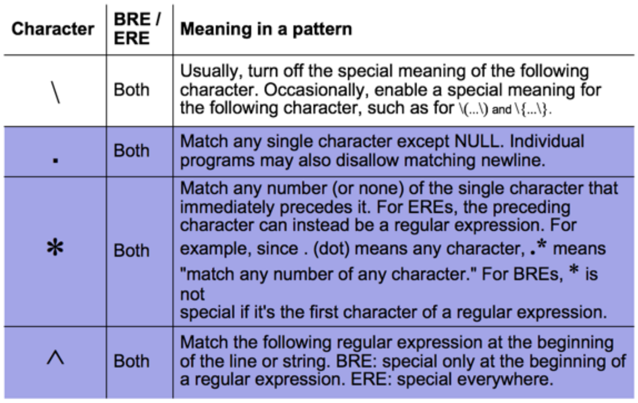
  - The "let" command is used to do arithmetic

- **For loop**

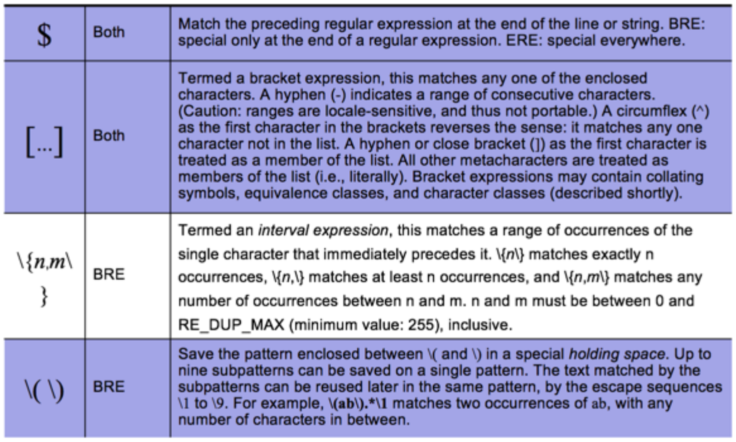
```
#!/bin/sh
temp=`ls`
for f in $temp
do
                echo $f
done
```

  - f will refer to each word in `ls` output

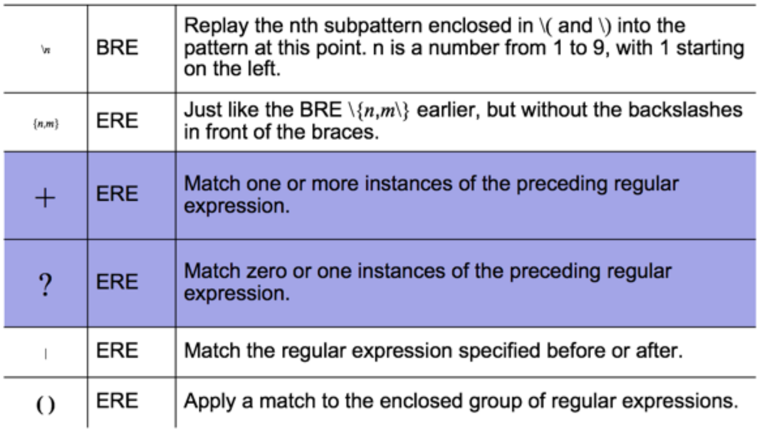




Character	BRE / ERE	Meaning in a pattern
\	Both	Usually, turn off the special meaning of the following character. Occasionally, enable a special meaning for the following character, such as for \(...\) and \{...\}.
.	Both	Match any single character except NULL. Individual programs may also disallow matching newline.
*	Both	Match any number (or none) of the single character that immediately precedes it. For EREs, the preceding character can instead be a regular expression. For example, since . (dot) means any character, .* means "match any number of any character." For BREs, * is not special if it's the first character of a regular expression.
^	Both	Match the following regular expression at the beginning of the line or string. BRE: special only at the beginning of a regular expression. ERE: special everywhere.



$	Both	Match the preceding regular expression at the end of the line or string. BRE: special only at the end of a regular expression. ERE: special everywhere.
[...]	Both	Termed a bracket expression, this matches any one of the enclosed characters. A hyphen (-) indicates a range of consecutive characters. (Caution: ranges are locale-sensitive, and thus not portable.) A circumflex (^) as the first character in the brackets reverses the sense: it matches any one character not in the list. A hyphen or close bracket (]) as the first character is treated as a member of the list. All other metacharacters are treated as members of the list (i.e., literally). Bracket expressions may contain collating symbols, equivalence classes, and character classes (described shortly).
\{*n*,*m*\ }	BRE	Termed an *interval expression*, this matches a range of occurrences of the single character that immediately precedes it. \{*n*\} matches exactly n occurrences, \{*n*,\} matches at least n occurrences, and \{*n*,*m*\} matches any number of occurrences between n and m. n and m must be between 0 and RE_DUP_MAX (minimum value: 255), inclusive.
\(\)	BRE	Save the pattern enclosed between \(and \) in a special *holding space*. Up to nine subpatterns can be saved on a single pattern. The text matched by the subpatterns can be reused later in the same pattern, by the escape sequences \1 to \9. For example, **\(ab\).*\1** matches two occurrences of ab, with any number of characters in between.



\n	BRE	Replay the nth subpattern enclosed in \(and \) into the pattern at this point. n is a number from 1 to 9, with 1 starting on the left.
{*n*,*m*}	ERE	Just like the BRE \{*n*,*m*\} earlier, but without the backslashes in front of the braces.
+	ERE	Match one or more instances of the preceding regular expression.
?	ERE	Match zero or one instances of the preceding regular expression.
|	ERE	Match the regular expression specified before or after.
( )	ERE	Apply a match to the enclosed group of regular expressions.

Matching Multiple Characters with One Expression

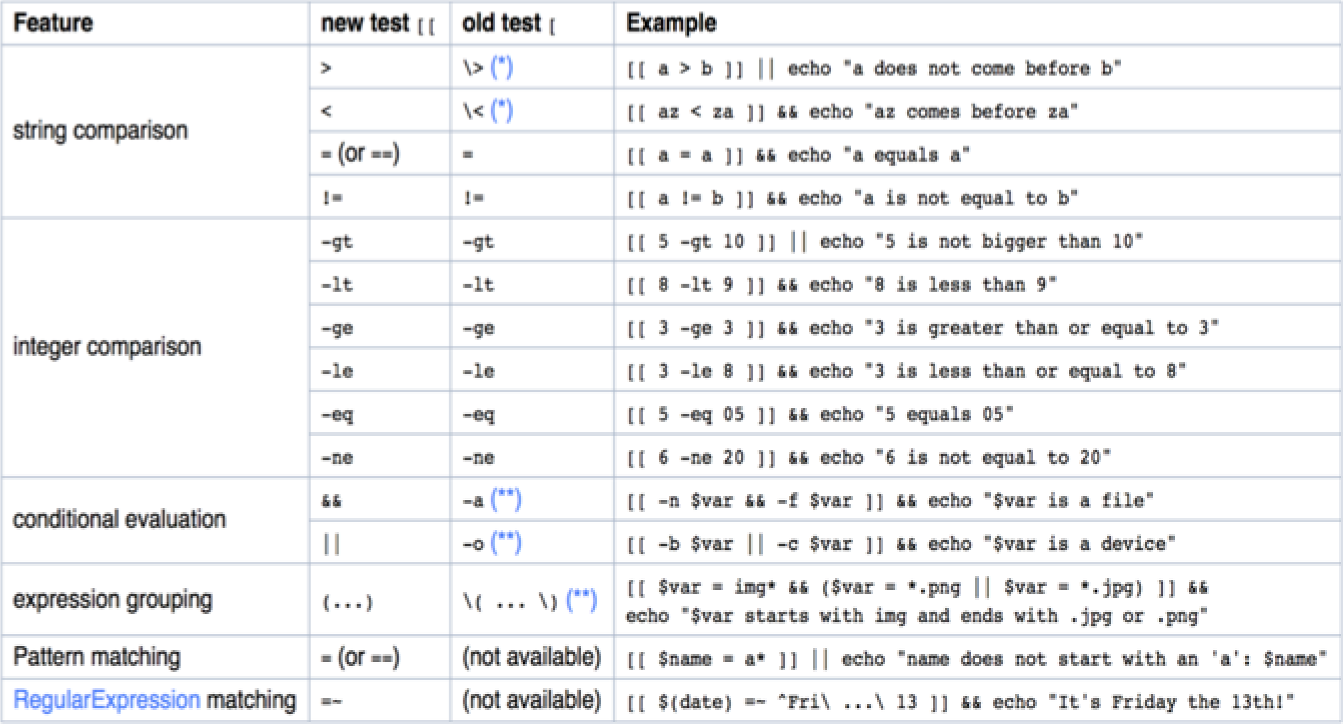
*	Match zero or more of the preceding character
\{*n*\}	Exactly n occurrences of the preceding regular expression
\{*n*,\ }	At least n occurrences of the preceding regular expression
\{*n*,*m*\ }	Between n and m occurrences of the preceding regular expression

POSIX Bracket Expressions

Class	Matching characters	Class	Matching characters
[:alnum:]	Alphanumeric characters	**[:lower:]**	**Lowercase characters**
[:alpha:]	Alphabetic characters	[:print:]	Printable characters
[:blank:]	**Space and tab characters**	[:punct:]	Punctuation characters
[:cntrl:]	Control characters	[:space:]	Whitespace characters
[:digit:]	Numeric characters	**[:upper:]**	**Uppercase characters**
[:graph:]	Nonspace characters	[:xdigit:]	Hexadecimal digits

Anchoring text matches

Pattern	Text matched (in bold) / Reason match fails
ABC	Characters 4, 5, and 6, in the middle: abc**ABC**defDEF
^ABC	Match is restricted to beginning of string
def	Characters 7, 8, and 9, in the middle: abcABC**def**DEF
def$	Match is restricted to end of string
[[:upper:]]\{3\}	Characters 4, 5, and 6, in the middle: abc**ABC**defDEF
[[:upper:]]\{3\}$	Characters 10, 11, and 12, at the end: abcDEFdef**DEF**
^[[:alpha:]]\{3\}	Characters 1, 2, and 3, at the beginning: **abc**ABCdefDEF



Feature	new test [[	old test [	Example
string comparison	`>`	`\>` (*)	`[[ a > b ]] || echo "a does not come before b"`
	`<`	`\<` (*)	`[[ az < za ]] && echo "az comes before za"`
	`=` (or `==`)	`=`	`[[ a = a ]] && echo "a equals a"`
	`!=`	`!=`	`[[ a != b ]] && echo "a is not equal to b"`
integer comparison	`-gt`	`-gt`	`[[ 5 -gt 10 ]] || echo "5 is not bigger than 10"`
	`-lt`	`-lt`	`[[ 8 -lt 9 ]] && echo "8 is less than 9"`
	`-ge`	`-ge`	`[[ 3 -ge 3 ]] && echo "3 is greater than or equal to 3"`
	`-le`	`-le`	`[[ 3 -le 8 ]] && echo "3 is less than or equal to 8"`
	`-eq`	`-eq`	`[[ 5 -eq 05 ]] && echo "5 equals 05"`
	`-ne`	`-ne`	`[[ 6 -ne 20 ]] && echo "6 is not equal to 20"`
conditional evaluation	`&&`	`-a` (**)	`[[ -n $var && -f $var ]] && echo "$var is a file"`
	`||`	`-o` (**)	`[[ -b $var || -c $var ]] && echo "$var is a device"`

expression grouping	`(...)`	`\( ... \)` (**)	`[[ $var = img* && ($var = *.png || $var = *.jpg) ]] &&`
			`echo "$var starts with img and ends with .jpg or .png"`

Pattern matching	`=` (or `==`)	(not available)	`[[ $name = a* ]] || echo "name does not start with an 'a': $name"`
RegularExpression matching	`=~`	(not available)	`[[ $(date) =~ ^Fri\ ...\ 13 ]] && echo "It's Friday the 13th!"`

WEEK 3

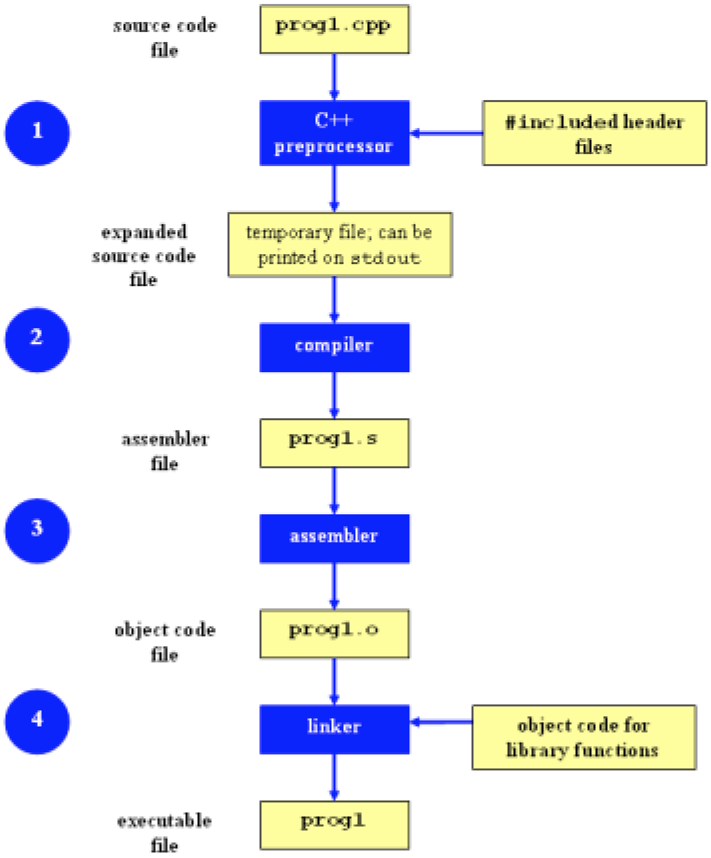
Decompressing Files

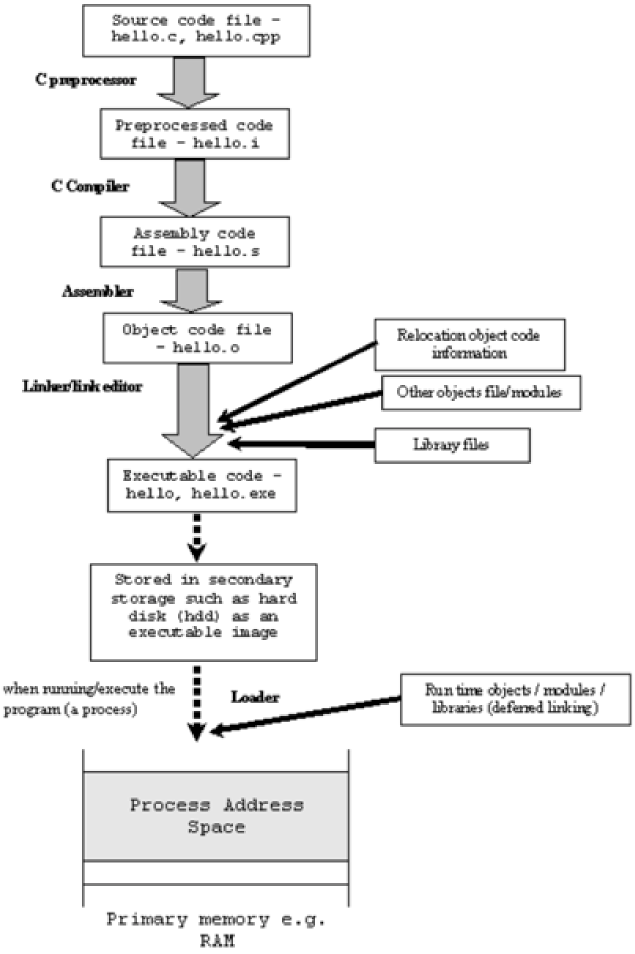
- $ tar -xzvf filename.tar.gz

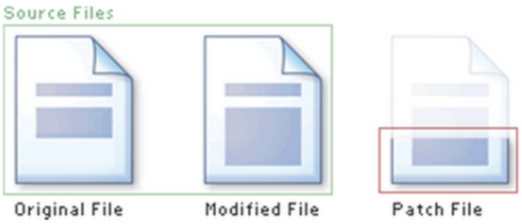
Compilation Process

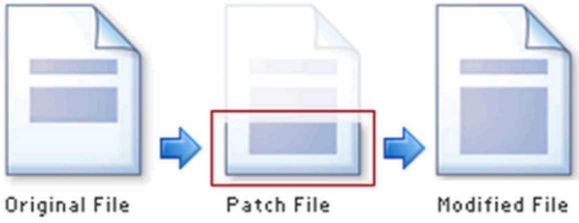
- source code -> **preprocessor** -> temp file -> **compiler** -> (.s) -> **assembler** -> (.o) -> **linker** -> executable
    - Preprocessing: source code stripped of comments. Anything with a # (macros, conditional compilation instructions, include headers) is resolved to their values and substituted into code
    - Compilation: preprocessed code converted into assembly .s file with ISA (instruction set architecture) of machine
    - Assembler: assembly code red and produces assembled into .o object file
    - Linking: object files and .so library files are combined into a single executable file. Resolves references to external symbols. (.exe)
- Command-Line Compilation
    - g++ -Wall file.cpp -o file
    - Example:
        - shop.cpp (#includes shoppingList.h and item.h)
        - shoppingList.cpp (#includes shoppingList.h)
        - item.cpp (#includes item.h)
        - **g++ -Wall shoppingList.cpp item.cpp shop.cpp –o shop**
    - Make change to source file: recompile code for each source file
        - Separate g++ command, less compiler work but more commands
    - Make change to header file: need to recompile every source file that includes it and every source file that includes a header that includes it
        - **Make**
            - Utility for managing large software projects, compiles files and keeps up to date, efficient compilation
            - Makefile example in pic, with makefile only changed files are 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 program code and creates executables in current temporary directory
            - --prefix flag
        - **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 dirs)
- **Patching**
    - Piece of software designed to fix problems with/update a program
    - A diff file including changes made to file
    - **diff Unified Format**

        - diff -u original_file modified_file
            - --- path/to/original_file
            - +++ path/to/modified_file
            - @@ -l,s +l,s @@
            - @@: beginning of a hunk
            - l: beginning line number
            - s: number of lines the change hunk applies to for each file
            - A line with a:
                - - sign was deleted from the original
                - + sign was added to the original
                -   stayed the same
    - patch -p*num* < patch_file
        - *num* is the number of slashes to remove from the path so that we apply the patch to that file
        - e.g. patch stuff in /usr/bin/stuff, do patch -p3
        - e.g. locally, do patch -p0
- Python
    - A Object-Oriented scripting language with classes and member functions
    - Compiled and interpreted (to bytecode => interpreted by Python interpreter)
    - Slower than C but easy to learn/read/use
    - Indentation: no braces/keywords, uses indentation/tabs/whitespace
    - Optparse Library
        - argument, option, option argument
    - **Python List**
        - Common data structure, like a C array but more **dynamic/mutable** and **heterogeneous**
            - Expands as new items added, can hold objects of different types
        - Access: list_name[index]
        - example:
            - t = [123, 3.0, ‘hello!’]
            - s = [2, 4, 7.0]
            - print t[0]
            - merge = t + s   //merge lists
    - **Python Dictionary**
        - Hash table, key-value pair storage with unique, immutable keys
        - dict = {}   // empty dictionary
        - example:
            - dict = {}
            - dict[‘hello’] = “world”
            - print dict[‘hello’]
            - del dict[‘hello’]
            - del dict
    - **For loops**
        - list = [‘Mary’, ‘had’, ‘a’]
        - for i in list:
              print i
        - for i in range(len(list)):
              print i




Source code file - hello.c, hello.cpp
C preprocessor
Preprocessed code file - hello.i
C Compiler
Assembly code file - hello.s
Assembler
Object code file - hello.o
Relocation object code information
Other objects file/modules
Linker/link editor
Library files
Executable code - hello, hello.exe
Stored in secondary storage such as hard disk (hdd) as an executable image
when running/execute the program (a process)
Loader
Run time objects / modules / libraries (deferred linking)
Process Address Space
Primary memory e.g. RAM





Makefile Example

```
# Makefile - A Basic Example
all : shop  #usually first
shop : item.o shoppingList.o shop.o
	g++ -g -Wall -o shop item.o shoppingList.o shop.o
item.o : item.cpp item.h
	g++ -g -Wall -c item.cpp
shoppingList.o : shoppingList.cpp shoppingList.h
	g++ -g -Wall -c shoppingList.cpp
shop.o : shop.cpp item.h shoppingList.h
	g++ -g -Wall -c shop.cpp
clean :
	rm -f item.o shoppingList.o shop.o shop
```

Rule

🟩 Comments
🟦 Targets
🟪 Prerequisites
🟫 Commands

Dependency Line

```
#!/usr/bin/python

import random, sys
from optparse import OptionParser

class randline:
         def __init__(self, filename):
                  f = open (filename,
'r')
                  self.lines =
f.readlines()
                  f.close ()

         def chooseline(self):
                  return
random.choice(self.lines)


def main():
    version_msg = "%prog 2.0"
    usage_msg = """%prog [OPTION]...
FILE Output randomly selected lines
from FILE."""

parser = OptionParser(version=version_msg,
                      usage=usage_msg)
parser.add_option("-n", "--numlines",
            action="store", dest="numlines",
            default=1, help="output NUMLINES
            lines (default 1)")

options, args = parser.parse_args(sys.argv[1:])

try:
    numlines = int(options.numlines)
except:
    parser.error("invalid NUMLINES: {0}".
                      format(options.numlines))
if numlines < 0:
    parser.error("negative count: {0}".
                 format(numlines))
if len(args) != 1:
    parser.error("wrong number of operands")
input_file = args[0]
try:
    generator = randline(input_file)
    for index in range(numlines):
        sys.stdout.write(generator.chooseline())
except IOError as (errno, strerror):
    parser.error("I/O error({0}): {1}".
format(errno, strerror))

if __name__ == "__main__":
    main()
```

Tells the shell which interpreter to use

Import statements, similar to include statements
Import OptionParser class from optparse module

The beginning of the class statement: randline
The constructor
Creates a file handle

Reads the file into a list of strings called lines
Close the file

The beginning of a function belonging to randline
Randomly select a number between 0 and the size of lines and returns the line corresponding to the randomly selected number

The beginning of main function

version message

usage message

Creates OptionParser instance

Start defining options, action "store" tells optparse to take next argument and store to the right destination which is "numlines". Set the default value of "numlines" to 1 and help message.

options: an object containing all option args

args: list of positional args leftover after parsing options

Try block
  get numline from options and convert to integer
Exception handling
  error message if numlines is not integer type, replace {0} w/ input
If numlines is negative
  error message
If length of args is not 1 (no file name or more than one file name)
  error message
Assign the first and only argument to variable input_file
Try block
  instantiate randline object with parameter input_file
  for loop, iterate from 0 to numlines – 1
    print the randomly chosen line
Exception handling
  error message in the format of "I/O error (errno):strerror

In order to make the Python file a standalone program

WEEK 4

- 2 ways to create version controlled project (git init, git clone)
git commands (commit, checkout, branch)
- why git add then git commit
- git branch is a pointer to a commit object
  ~ default branch is Master, moves each time a commit is added

Software Development Process

- Lots of code changes, new features, bug fixes, performance enhancements
- Many people working on same/different parts of code
- Need for versioning, especially with users who use different versions

Source/Version Control

- Track changes to code/files, history of software (Git, Subversion, Perforce)
- **Local VCS**: different versions as folders in local machine, no server, users copy via disk/network
- **Centralized VCS**: versions sit on central server, users get working copy of files, changes committed to server, all users can get changes
- **Distributed VCS**: version history replicated at every user's machine, users have version control all of the time, changes can be communicated between users, GIT

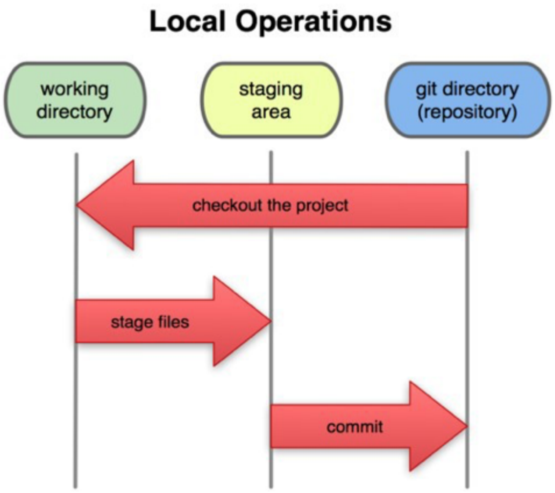
Key Terms

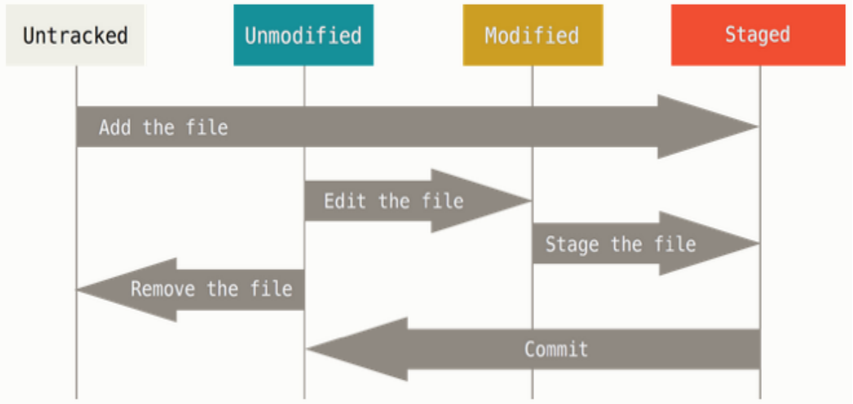
- **Repository**: files and folders related to software code, full history of software
- **Working copy:** copy of software files in the repository
- **Checkout**: create a working copy of the repository
- **Checkin/Commit**: write changes made in working copy to repository, recorded by VCS

Git Source Control

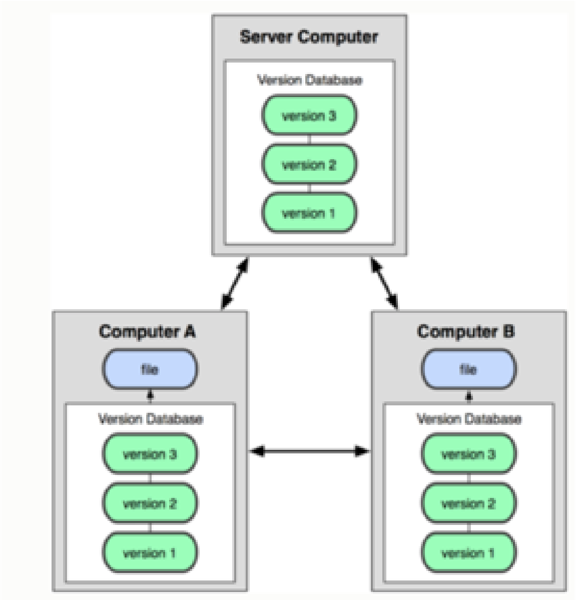
- **Repository Objects** used by GIT to implement source control
  - **Blobs**: sequences of bytes
  - **Trees:** groups blobs/trees together
  - **Commit**: refers to a particular "git commit", contains all info about commit
  - **Tags**: a named commit object for convenience (e.g. versions of the software)
- Objects are uniquely identified with **hashes**
- **Git States**
  - Working directory -- git add --> staging area -- git commit --> git directory/repository -- git checkout --> working directory
- **Git Terms**
  - **Head:** refers to commit object, there can be many heads in a repository
  - **HEAD:** refers to currently active head
  - **Detached HEAD:** if commit is not pointed to by a branch, no commits can be preserved unless a branch is created (git checkout v3.0 -b BranchV3.1)
  - **Branch:** refers to a head and its entire set of ancestor commits
  - **Master:** default branch
- **Creating Git Repository**
  - mkdir gitroot; cd gitroot
  - git init // creates empty git repo
  - touch hello.txt
  - git add .
  - git commit -m "yo yo yo"

- **Git commands**
    - git clone: create copy of existing repo
    - git checkout <tag/commit> -b <new_name>
    - git status: shows list of modified files
    - git diff: compares working copy with staged files
    - git log: shows history of commits
    - git show: show a certain object in the repo
    - git add <file>
    - git commit
    - git diff HEAD: see changes in working version
    - git help
    - git pull: incorporate changes from remote repo to current branch
    - Reverting
        - git checkout HEAD main.cpp: gets HEAD revision for working copy
        - git checkout -- main.cpp: reverts changes in working directory
        - git revert: reverts commits (creating new commits)
        - git reset: deletes commit completely, cleaner
    - Cleaning up untracked files
        - git clean
    - Tagging: human readable pointers to specific commits
        - git tag -a v1.0 -m ‘Version 1.0’
        - Names the HEAD commit as v1.0
- **Git integrating changes**
    - Changes in multiple branches, can merge and rebase
    - merge is simpler/straightforward, rebase is cleaner
- Branch to experiment with code without affecting main branch, separate projects with common code base, multiple versions of the same project

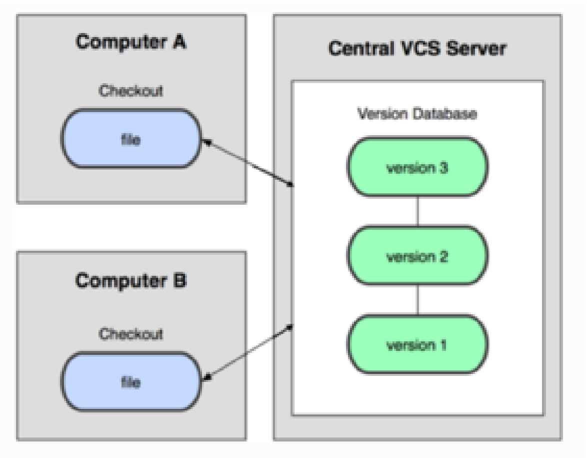


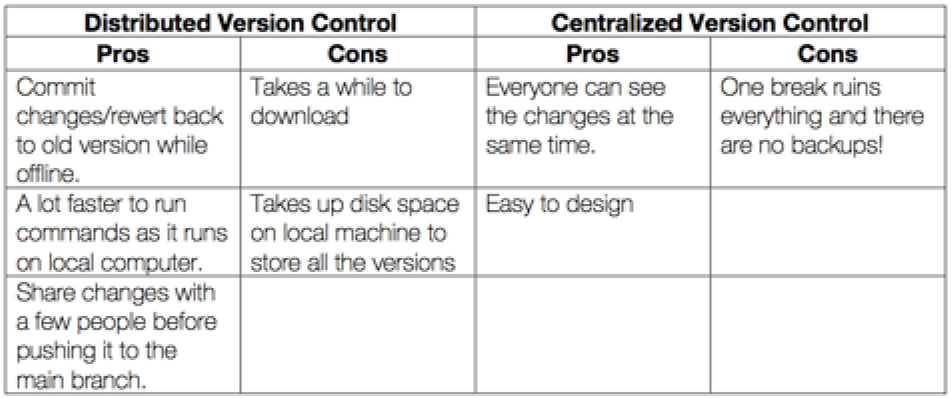


Distributed



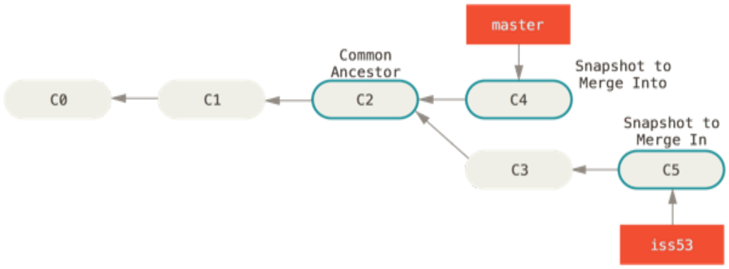
Centralized

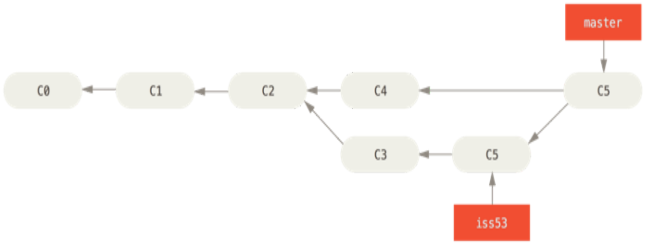




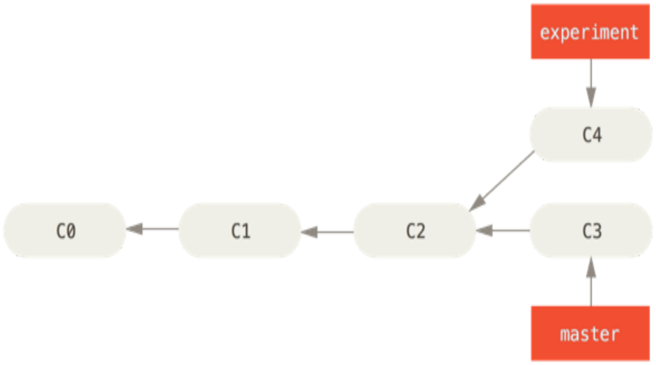
Distributed Version Control		Centralized Version Control	
Pros	Cons	Pros	Cons
Commit changes/revert back to old version while offline.	Takes a while to download	Everyone can see the changes at the same time.	One break ruins everything and there are no backups!
A lot faster to run commands as it runs on local computer.	Takes up disk space on local machine to store all the versions	Easy to design	
Share changes with a few people before pushing it to the main branch.			

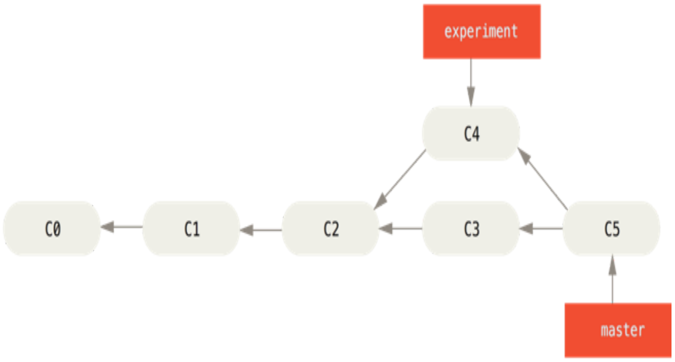
Merge

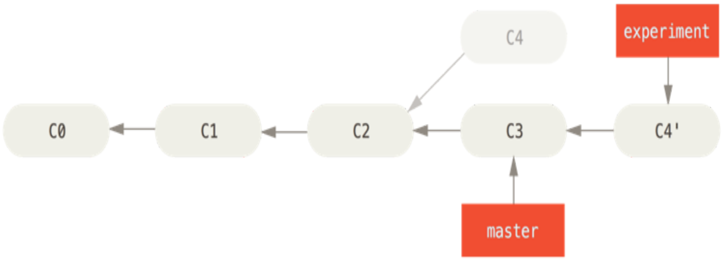


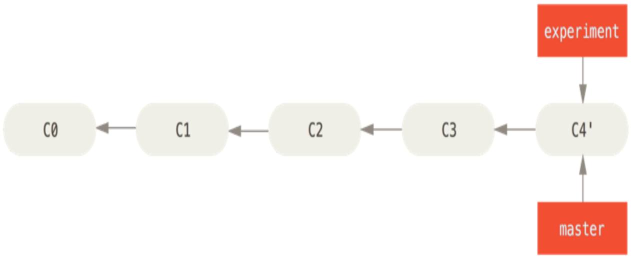


Rebase









WEEK 5

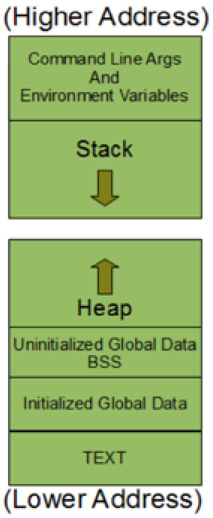
Debugging Process

- Reproduce bug, simplify program input, debugger to track origin of problem, fix
- Debugger: program used to run and debug other target programs
    - Advantages:
        - Can step through source code line by line
        - Each line executed on demand
        - Interact with and inspect program at run time
        - If program crashes, debugger outputs where and why
- GDB - GNU debugger for several languages
    - **Compile program**
        - Normally: $ gcc [flags] <source files> -o <output file>
        - Debugging: $ gcc [other flags] **-g** <source files> -o <output file>
            - enables built-in debugging support
    - **Specify Program to Debug**
        - $ gdb <executable>
        - $ gdb
          (gdb) file <executable>
    - **Run Program**
        - (gdb) run
        - (gdb) run [arguments]
    - **Exit the gdb Debugger**
        - (gdb) quit
- **Run-time Errors**
    - **Segmentation Fault** (SIGSEGV)
        - Code error
    - **Logic Error**
        - Successfully runs and exits
- **Breakpoints**
    - Used to stop the running program at a specific point
    - Pauses and prompts for command
        - (gdb) break file1.c:6
            - Pauses at line 6 of file1.c
        - (gdb) break my_function
            - Pauses at first line of my_function every time it's called
        - (gdb) break [position] if <expression>
            - Only pauses as position when expression evaluates to true
    - (gdb) info breakpoints //break|br|b
    - If no arguments are provided to the below commands, all breakpoints are affected!! (delete, disable, enable)
    - (gdb) delete [bp_number | range]
        - Deletes the specified breakpoint or range of breakpoints
    - (gdb) disable [*bp_number* | *range*]
        - Temporarily deactivates a breakpoint or a range of breakpoints
    - (gdb) enable [*bp_number* | *range*]
        - Restores disabled breakpoints
    - (gdb) ignore *bp_number iterations*

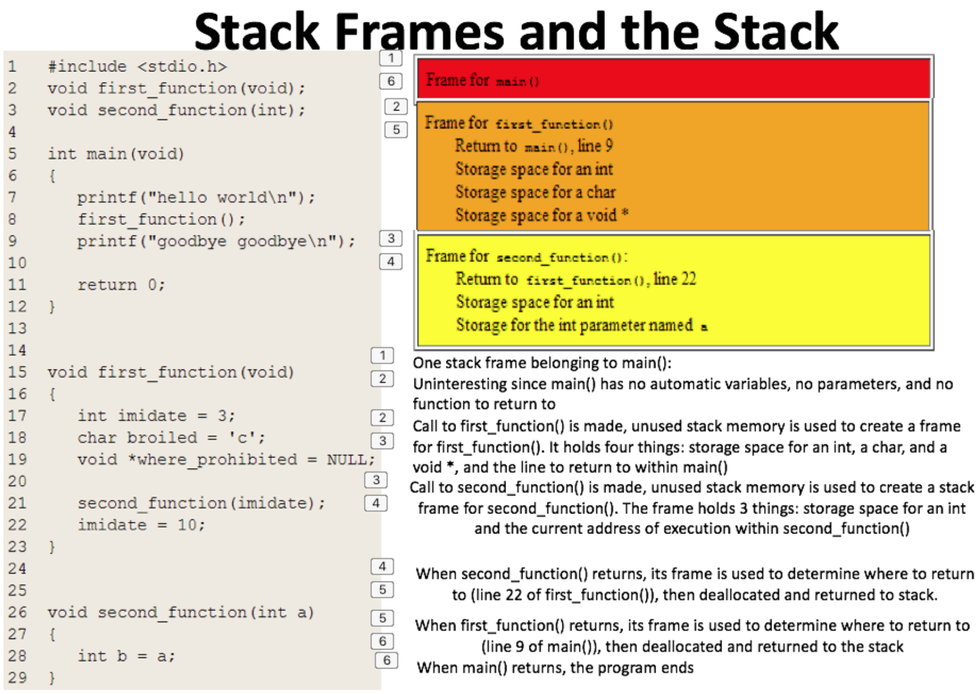
        - Instructs GDB to pass over a breakpoint without stopping a certain number of times.
            - bp_number: the number of a breakpoint
            - Iterations: the number of times you want it to be passed over
- **Displaying Data**
    - Interrupt execution with breakpoint, then print data of interest
    - (gdb) print [/format] <expression>
        - prints the value of the specified expression in the specified format
        - d: decimal
        - x: hexadecimal
        - o: octal
        - t: binary
- **Resuming Execution after Break**
    - 4 options, c/s/n/f
    - **c or continue**: debugger will continue executing until next breakpoint
    - **s or step**: debugger will continue to next source line, steps IN to fcns
    - **n or next**: debugger will continue to next source line in the current (innermost) stack frame
    - **f or finish**: debugger will resume execution until the current function returns. Execution stops immediately after the program flow returns to the function's caller
        - the function's return value and the line containing the next statement are displayed
- **Watchpoints**
    - Watch/observe changes to variables
    - (gdb) watch my_var
        - debugger will stop the program when the value of my_var changes
        - prints old and new values
    - (gdb) rwatch <expression>
        - debugger stops the program whenever the program reads the value of any object involved in the evaluation of *expression*
- **Process Memory Layout**
    - In pic
    - Stack info
        - Program made up of one or more functions that interact by calling each other
        - Each time a function is called, an area of memory is set aside for it
            - This area of memory is called a **stack frame**
            - Holds info
                - storage space for local vars
                - memory address to return to when called function returns
                - arguments/parameters of called function
            - Each function call gets its own stack frame
        - Collectively all stack frames compose the **call stack**
            - Function call -> stack frame created
            - Function returns -> stack frame deallocation
        - Analyze stack in GDB
            - (gdb) backtrace   //bt

  - Shows the call trace (call stack)
  - Each frame listing gives the arguments to that function and the line number currently being executed within that frame
- (gdb) info frame
  - Displays information about the current stack frame, including its return address and saved register values
- (gdb) info locals
  - Lists the local variables of the function corresponding to the stack frame, with their current values
- (gdb) info args
  - List the argument values of the corresponding function call
- (gdb) info functions
  - List all functions in the program
- (gdb) list

Lists all source code lines around the current line


(Higher Address)
Command Line Args And Environment Variables
Stack
Heap
Uninitialized Global Data BSS
Initialized Global Data
TEXT
(Lower Address)

- TEXT segment
  - Contains machine instructions to be executed
- Global Variables
  - Initialized
  - Uninitialized
- Heap segment
  - Dynamic memory allocation
  - malloc, free
- Stack segment
  - Push frame: Function invoked
  - Pop frame: Function returned
  - Stores
    - Local variables
    - Return address, registers, etc
- Command Line arguments and Environment Variables



Stack Frames and the Stack

```
1    #include <stdio.h>
2    void first_function(void);
3    void second_function(int);
4
5    int main(void)
6    {
7       printf("hello world\n");
8       first_function();
9       printf("goodbye goodbye\n");
10
11      return 0;
12   }
13
14
15   void first_function(void)
16   {
17      int imidate = 3;
18      char broiled = 'c';
19      void *where_prohibited = NULL;
20
21      second_function(imidate);
22      imidate = 10;
23   }
24
25
26   void second_function(int a)
27   {
28      int b = a;
29   }
```

Frame for main()

Frame for first_function()
Return to main(), line 9
Storage space for an int
Storage space for a char
Storage space for a void *

Frame for second_function():
Return to first_function(), line 22
Storage space for an int
Storage for the int parameter named a

One stack frame belonging to main():
Uninteresting since main() has no automatic variables, no parameters, and no function to return to

Call to first_function() is made, unused stack memory is used to create a frame for first_function(). It holds four things: storage space for an int, a char, and a void *, and the line to return to within main()

Call to second_function() is made, unused stack memory is used to create a stack frame for second_function(). The frame holds 3 things: storage space for an int and the current address of execution within second_function()

When second_function() returns, its frame is used to determine where to return to (line 22 of first_function()), then deallocated and returned to stack.

When first_function() returns, its frame is used to determine where to return to (line 9 of main()), then deallocated and returned to the stack

When main() returns, the program ends

C Programming

Basic Data Types

- int: Holds integer numbers (4 bytes)
- float: Holds floating point numbers (4 bytes)
- double: Holds higher-precision floating point numbers (8 bytes)
- char: Holds a byte of data, characters
- void
- NO bool before C99

Pointers

- Variables that store memory addresses
- **Declaration**: <variable_type> *<name>;

  -
    ```
    int *ptr;
    int var = 77;
    ptr = &var;
    ```
- **Dereferencing**: access value that pointer points to

  -
    ```
    double x, *ptr;
    ptr = &x;
    *ptr = 7.8;       //assigns value 7.8 to x
    ```
- **Pointers to Pointers**

  -
    ```
    char c = ‘A’;
    char *cPtr = &c;
    char **cPtrPtr = &cPtr
    // cPtrPtr -> cPtr -> c
    //  &cPtr      &c    ‘A’
    ```
- **Pointers to Functions**, aka function pointers or functors
  - Pass a function to another, like a sort function

  -
    ```
    Declaration: double (*func_ptr)(double, double);
                 func_ptr = [&]pow;
    ```
  - Usage:
    - double result = (*func_ptr)(1.5, 2.0);
      - call the function referenced by func_ptr
    - result = func_ptr(1.5, 2.0);
      - same as a function call
  - One case is a function that returns a function pointer
    - (return type of function pointed to) (*func(input params))(what function it is pointing to would take in)
    - int (*(foo(int i)) (int);

Reading/Writing Chars

- **int getchar();**
  - Returns next char from stdin
  - Allows for buffered I/O, store info and then make fewer syscalls later
- **int putchar(int char);**
  - writes a character to current position in stdout
  - Allows for buffered I/O

Formatted I/O

- int fprintf(FILE *fp, const char * format, …);
- int fscanf(FILE *fp, const char * format, …);
- FILE *fp can be either a file pointer or stdin, stdout, stderr
- Format string

  -
    ```
    int score = 120; char player[] = “Mary”;
    fp = fopen(“file.txt”, “w+”)
    fprintf(fp, “%s has %d points.\n”, player, score);
    ```

Structs

- No classes in C
- Used to package related data, vars of different types, together

  ```
  struct Student {
        char name[64];
        char UID[10];
        int age;
  };
  ```
- C Structs vs. C++ Classes
  - C: no member functions, no access specifiers, no defined constructors
  - C++: can have member functions, access specifiers (private by default), must have at least a default constructor

Dynamic Memory

- Memory allocated at runtime on the heap
  - Heap: can deallocate/allocate at will, not restricted to any order
  - Stack: LIFO, particular order, less flexible
  - void *malloc (size_t size);
    - Returns a pointer to the allocated memory of size bytes
  - void *realloc (void *ptr, size_t size);
    - Changes size of memory block pointed to by ptr to size bytes
  - void **free** (void *ptr);
    - Frees the block of memory pointed to by ptr

WEEK 6

Processor Modes

- Operating modes that place restrictions on the type of operations that can be performed. Switch between modes with system calls
  - User mode: restricted access to system resources (memory, I/O, CPU)
    - CPU restricted, specified area of memory
  - Kernel/supervisor mode: unrestricted
    - Unrestricted CPU, can use all instructions, access to all areas of memory, can take over CPU anytime
  - Dual-Mode makes sure system resources are shared among processes
    - Ensures **protection** from malicious programs and **fairness**
  - Goals for Protection
    - I/O, Memory, CPU
    - Prevent illegal operations, illegal access, modifying data, using too much CPU for too long
    - Instructions that interfere with goals are "privileged" and can only be executed by trusted code (the KERNEL)
      - Kernel managed hardware resources (CPU, Memory, I/O)
      - System call interface is safe way to expose privileged functions
      - Kernel executed privileged operations on behalf of user processes

System Calls

- Type of function used by user-level processes to request service from kernel
- Changes CPU's mode from user to kernel mode to enable capabilities
- Verified user is allowed to do action then performs operation on behalf of user
- The only way for a user to perform privileged operations/access system resources
- System call -> program being executed is interrupted -> control passed to kernel
- EXPENSIVE system call overhead
  - System must interrupt process, save state of process
  - OS takes control of CPU, verifies validity of operation
  - OS performs requested action
  - OS restored saved state/context, switches to user mode, gives control of CPU to user process
- System Calls
  - #include <unistd.h>
  - ssize_t read(int fildes, void *bug, size_t nbyte)
    - fildes = file descriptor (0 stdin, 1 stdout, 2, stderr)
    - buf = buffer to write to
    - nbyte = number of bytes to read
  - ssize_t write(int fildes, const void *buf, size_t nbyte);
  - int open(const char *pathname, int flags, mode_t mode);
  - int close(int fd);
  - pid_t getpid(void)
    - returns the process ID of calling process
  - int dup (int fd)
    - Duplicates a file descriptor fd, returns a second file descriptor that points to the same file table entry as fd does
  - int fstat(int filedes, struct stat *buf)
    - Returns info about the file with the descriptor filedes into buf

Library Functions

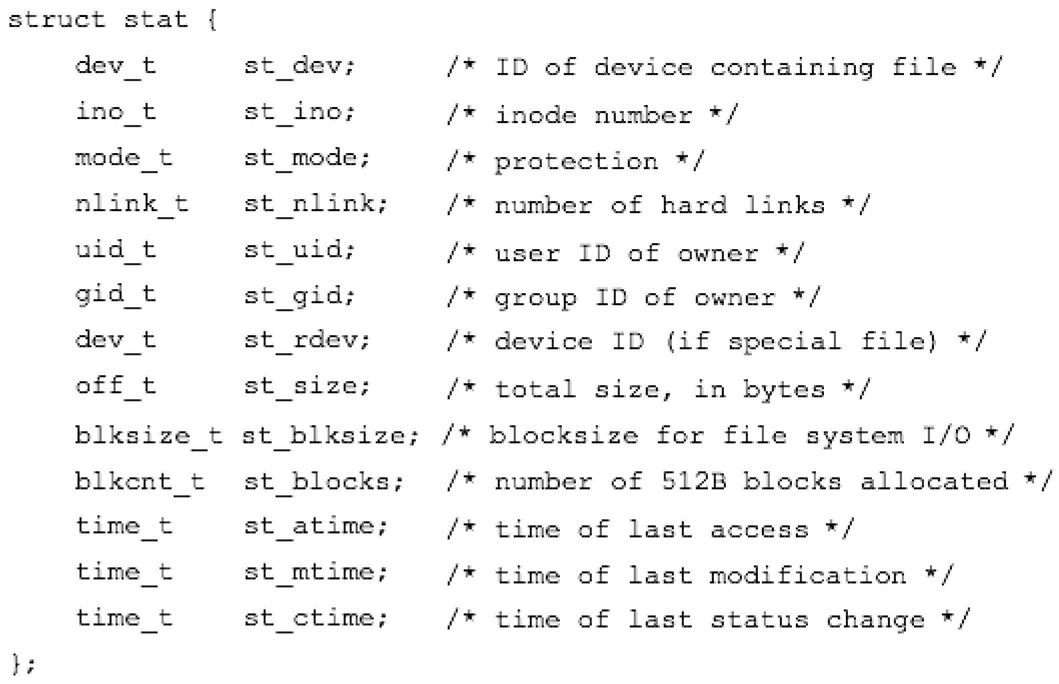
- Part of standard C library
- Avoid system call overhead, use equivalent library functions
  - **getchar**::read, **putchar**::write (standard I/O)
  - **fopen**::open, **fclose**::close (file I/O)
- These functions perform privileged operations via system calls, but do so indirectly. Use the same system calls but much fewer
  - Less switches between user and kernel mode is less overhead

Unbuffered vs. Buffered I/O

- Unbuffered: every byte is read/written by kernel through syscall
- Buffered: collect as many bytes as possible in a buffer and read more than a single byte into buffer at a time… then use one syscall for block of bytes
  - Decreases the number of read/write syscalls and thus overhead
  - Risk losing buffer (say power outage, or file modified before buffer written)

Commands

- **time**: outputs real, user, sys times
  - Real: read by clock
  - User: CPU time used by process
  - Sys: CPU time used by system on behalf of process
- **strace**: intercepts and prints out syscalls to stderr or to an output file



```
struct stat {
    dev_t     st_dev;     /* ID of device containing file */
    ino_t     st_ino;     /* inode number */
    mode_t    st_mode;    /* protection */
    nlink_t   st_nlink;   /* number of hard links */
    uid_t     st_uid;     /* user ID of owner */
    gid_t     st_gid;     /* group ID of owner */
    dev_t     st_rdev;    /* device ID (if special file) */
    off_t     st_size;    /* total size, in bytes */
    blksize_t st_blksize; /* blocksize for file system I/O */
    blkcnt_t  st_blocks;  /* number of 512B blocks allocated */
    time_t    st_atime;   /* time of last access */
    time_t    st_mtime;   /* time of last modification */
    time_t    st_ctime;   /* time of last status change */
};
```

WEEK 7

Parallelism

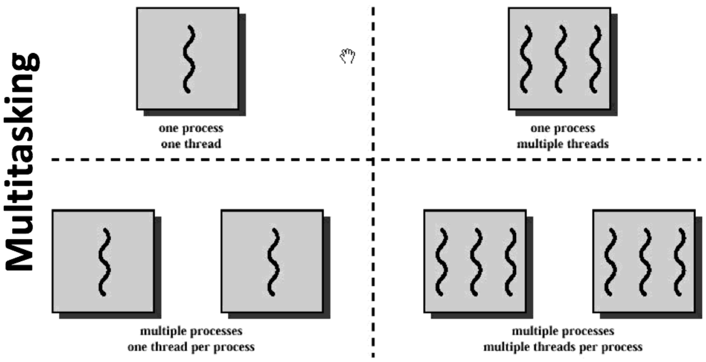
- **Multiprocessing:** use of multiple CPUs/cores to run multiple tasks simultaneously
- **Multitasking:** multiple processes schedules alternatively or maybe simultaneously on multiprocessing system
  - cmd1 | cmd2 | cmd 3
  - 3 different processes with own address space, communicates through pipes/syscalls
- **Multithreading:** same job broken into pieces/threads which may be executed simultaneously on multiprocessing system
  - **Thread:** Flow of instructions, path of execution within a process, smallest unit of processing scheduled by OS.
    - Multiple threads can be run on a uniprocessor (switches between different threads) or multiprocessor (runs at same time)
  - Threads share process's memory except for stacks, no extra work for data sharing
- Multiple processes or multiple threads
  - Really complex processes with multiple threads will slow it down, less resources allocated to each thread
- **Shared Memory in Multithreading**
  - Powerful => easy data access and sharing
  - Efficient => no need for syscalls, less expensive thread creation/destruction than process creation/destruction
  - Non-trivial => prevent several threads from accessing and changing the same shared data at the same time (**synchronization**)
    - **Race condition**: result depends on order of execution, synchronization needed, data must be accessed in particular order
    - **Atomicity:** synchronization used to make sure an operation is not splittable into parts, it is atomic
      - Any other thread will see the atomic operation before or after modification, but never the intermediate value between read/write

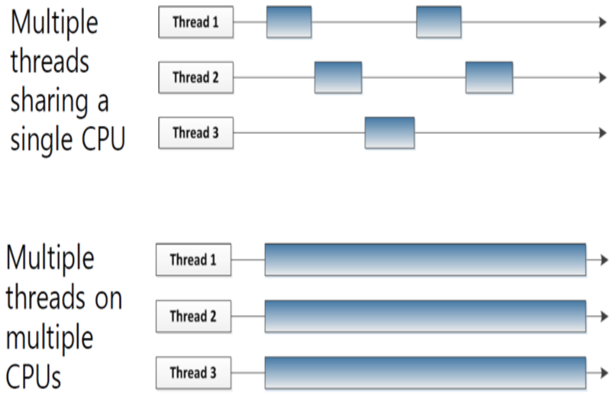
Pthread Library

- **pthread_create:** creates a new thread within a process and makes executable
  - Can be called any number of times from anywhere within code
  - returns 0 for success, else error number
  - int pthread_create(pthread_t *tid, const pthread_att_t *attr, void *(my_function)(void *), void *arg);
    - tid = unique identifier for newly created thread
    - attr = object that holds thread attributes (NULL for default)
    - my_function = function that thread will execute once created
    - arg = single argument that may be passed to my_function (NULL if none)
- **pthread_join:** waits for another thread to terminate
  - Makes originating thread wait for completion of all spawned threads
  - Prevents abortion of child threads before completion
  - returns 0 for success, else error number
  - int pthread_join(pthread_t tid, void **status);
    - tid = thread ID of thread to wait on
    - status = exit status of target thread stored in location pointed to by *status. Pass in NULL if status not needed

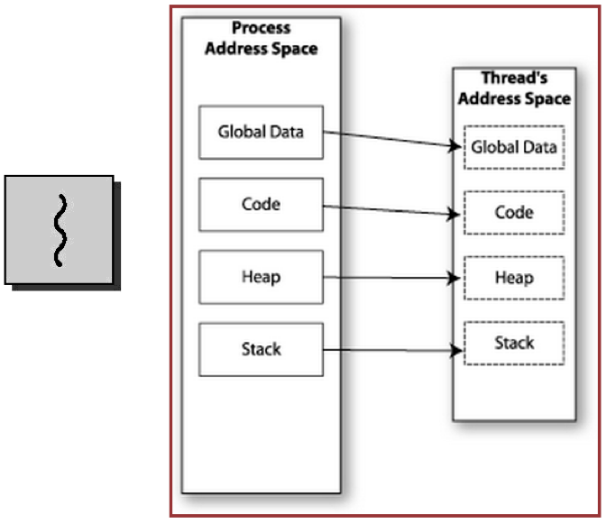
- **pthread_equal:** compares thread ids to see if they refer to the same thread
- **pthread_self:** returns the id of the calling thread
- **pthread_exit:** terminates the currently running thread

Multithreading

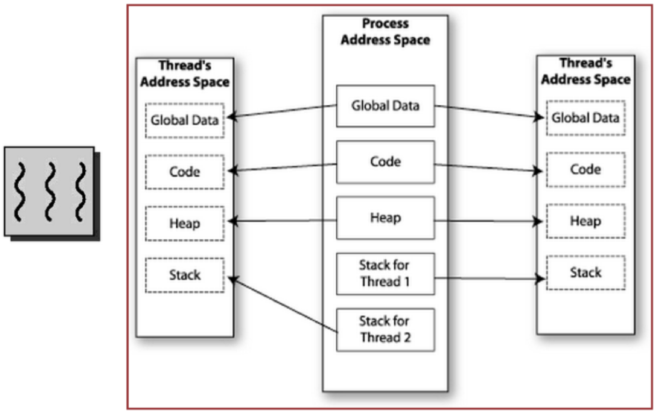



Multiple threads sharing a single CPU
Thread 1
Thread 2
Thread 3
Multiple threads on multiple CPUs
Thread 1
Thread 2
Thread 3

Memory Layout: Single-Threaded Program



Memory Layout: Multithreaded Program



Multithreading & Multitasking: Comparison

- **Multithreading**
  - Threads share the same address space
    - Light-weight creation/destruction
    - Easy inter-thread communication
    - An error in one thread can bring down all threads in process
- **Multitasking**
  - Processes are insulated from each other
    - Expensive creation/destruction
    - Expensive IPC
    - An error in one process cannot bring down another process

Parameters

```
int pthread_create( pthread_t *tid, const pthread_attr_t *attr,
                    void *(my_function)(void *), void *arg );
```

- **tid**: unique identifier for newly created thread
- **attr**: object that holds thread attributes (priority, stack size, etc.)
  - Pass in NULL for default attributes
- **my_function**: function that thread will execute once it is created
- **arg**: a *single* argument that may be passed to my_function
  - Pass in NULL if no arguments

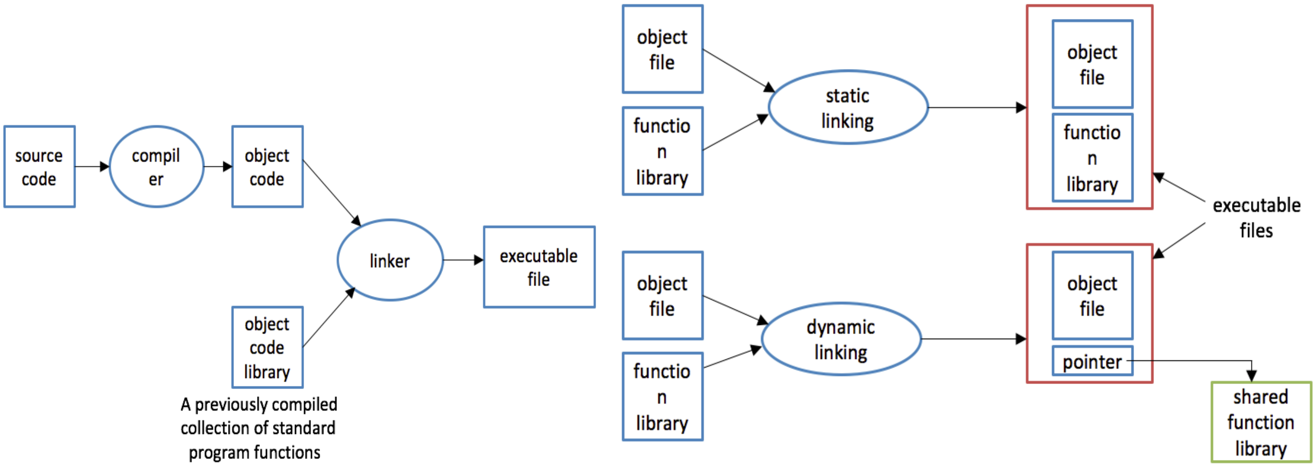
WEEK 8

Creating an Executable

- Source code -> compiler -> object code -> linker -> executable file
- Compiler translates programming language into machine language
- Linker takes 1+ object files and combines into single executable file

Static Linking

- Carries out once to produce single executable file
- If static libraries are called, linker copies all referenced modules into executable
- Typically denoted with .a extension for static libraries



Dynamic Linking

- Allows process to add/remove/replace/relocate object modules during execution
- If shared libraries are called
  - Only copies some reference info when executable file is created
  - Complete linking occurs during loading/running time
- Denoted with .so file extension (or .dll on Windows)
- Code typically compiled as *dynamic shared object* (DSO), default linking
- Pros/cons:
  - Size of dynamically linked executables are much much smaller
  - Takes longer to run, but faster to compile
  - Bit of a performance hit
    - Need to load shared objects at runtime at least once
    - Need to resolve addresses once or every time
  - Easier to update/modify
  - Bad if libraries have changed/tampered with
  - Usually if library change, code that references it does not need to be recompiled
- RTLD_LAZY vs RTLD_NOW: Resolved later vs resolved now

Linking and Loading

- Linker collects procedures and links together object modules into one executable program
- Why not save necessity of linking and write as one big program
  - Efficiency (one fcn changed, recompile entire thing vs recompile one function and link)
  - Multi-language programs

- Default linking is dynamic DSO
  - gcc -static hello.c -o hello-static
  - gcc hello.c -o hello-dynamic

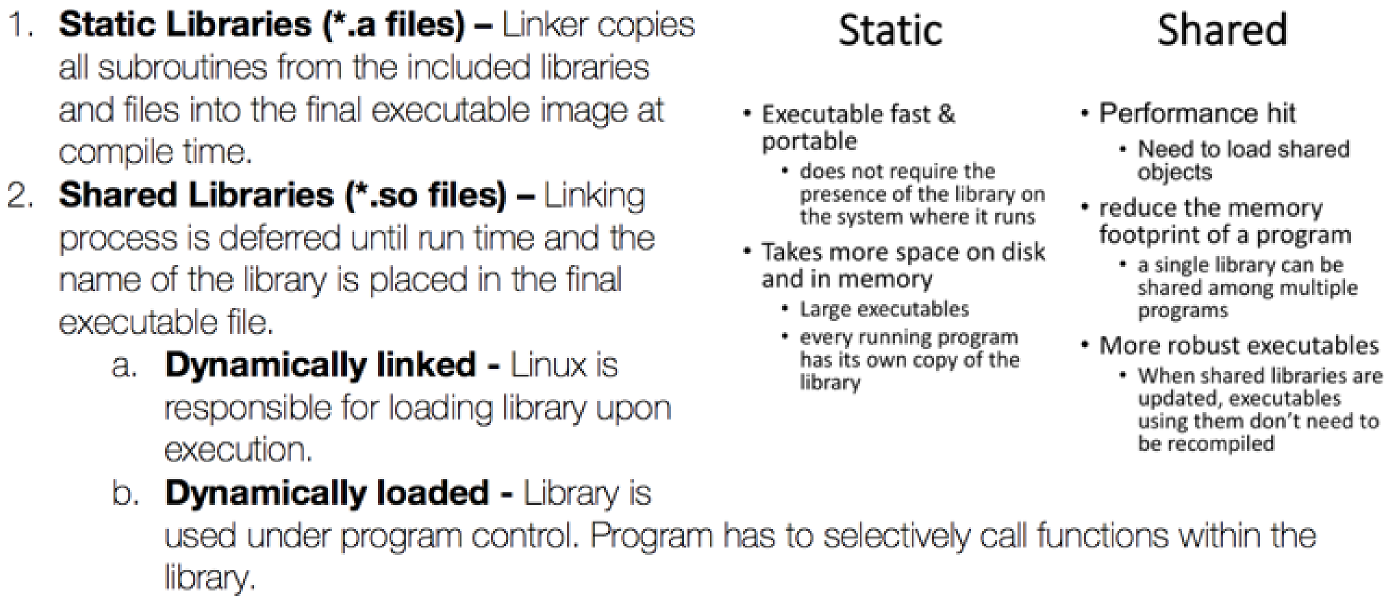
GCC Flags

- **-fPIC**: Compiler directive to output position independent code, a characteristic required by shared libraries.
- **-l*library***: Link with "lib*library*.a”
- Without -L to directly specify the path, /usr/lib is used.
- **-L**: At **compile** time, find the library from this path.
- **-Wl,rpath=.: -Wl** passes options to linker. **-rpath** at **runtime** finds .so from this path.
- **-c**: Generate object code from c code.
- **-shared**: Produce a shared object which can then be linked with other objects to form an executable.

Attributes of Functions

- Used to declare certain things about functions called in program
  - Helps compiler optimize calls and check code
- Used to control memory placement, code generation options or call/return conventions
- Introduced by ***attribute*** keyword on a declaration, followed by an attribute specification inside double parentheses

```
__attribute__ ((__constructor__))   //Is run when dlopen() is called
__attribute__ ((__destructor__))    //Is run when dlclose() is called
Example:
__attribute__ ((__constructor__))
void to_run_before (void) {
  printf("pre_func\n");
}
```



1. **Static Libraries (*.a files) –** Linker copies all subroutines from the included libraries and files into the final executable image at compile time.
2. **Shared Libraries (*.so files) –** Linking process is deferred until run time and the name of the library is placed in the final executable file.
   a. **Dynamically linked -** Linux is responsible for loading library upon execution.
   b. **Dynamically loaded -** Library is used under program control. Program has to selectively call functions within the library.

Static

- Executable fast & portable
  - does not require the presence of the library on the system where it runs
- Takes more space on disk and in memory
  - Large executables
  - every running program has its own copy of the library

Shared

- Performance hit
  - Need to load shared objects
- reduce the memory footprint of a program
  - a single library can be shared among multiple programs
- More robust executables
  - When shared libraries are updated, executables using them don’t need to be recompiled

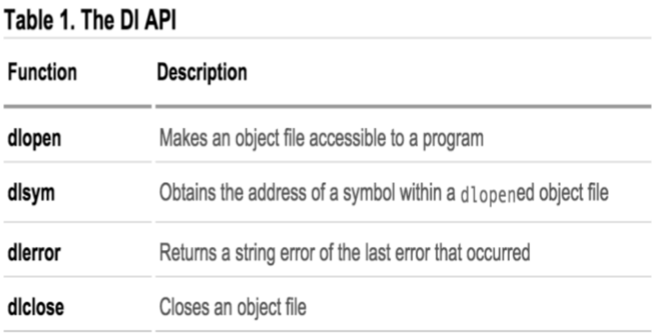
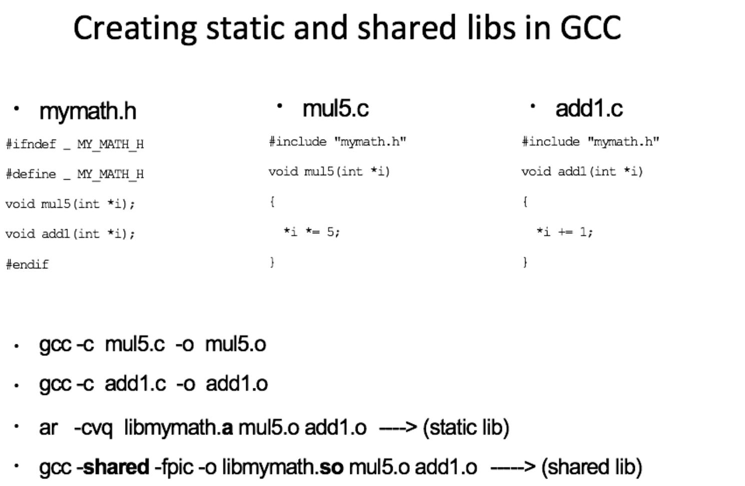


Table 1. The Dl API

Function	Description
dlopen	Makes an object file accessible to a program
dlsym	Obtains the address of a symbol within a `dlopen`ed object file
dlerror	Returns a string error of the last error that occurred
dlclose	Closes an object file



Creating static and shared libs in GCC

- mymath.h

```
#ifndef _ MY_MATH_H
#define _ MY_MATH_H
void mul5(int *i);
void add1(int *i);
#endif
```

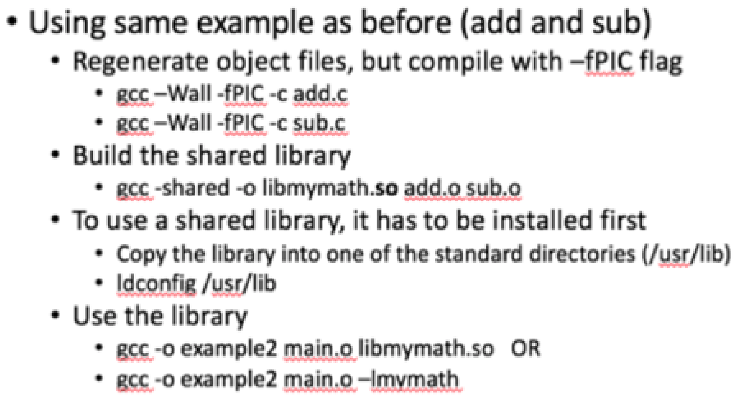
- mul5.c

```
#include "mymath.h"
void mul5(int *i)
{
  *i *= 5;
}
```

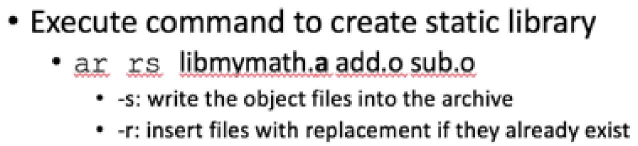
- add1.c

```
#include "mymath.h"
void add1(int *i)
{
  *i += 1;
}
```

- gcc -c mul5.c -o mul5.o
- gcc -c add1.c -o add1.o
- ar -cvq **libmymath.a** mul5.o add1.o ---> (static lib)
- gcc -**shared** -fpic -o libmymath.**so** mul5.o add1.o ----> (shared lib)

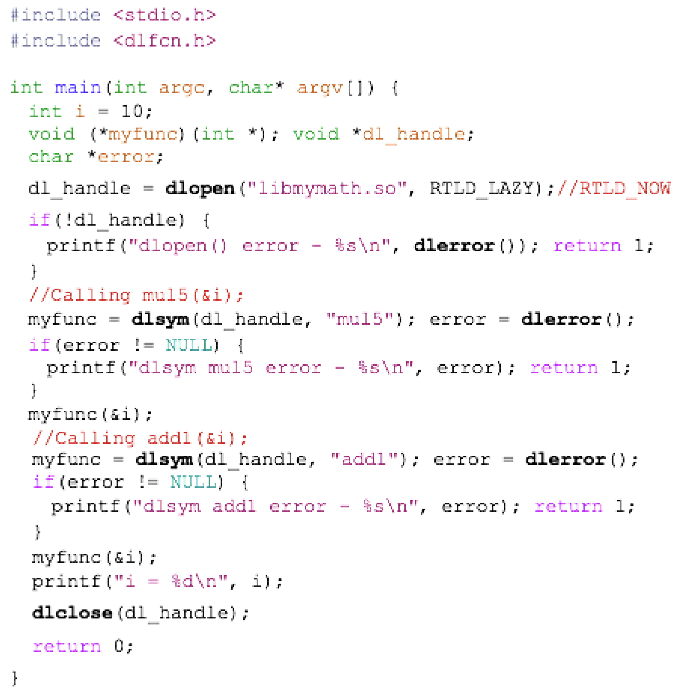


- Using same example as before (add and sub)
  - Regenerate object files, but compile with –fPIC flag
    - gcc –Wall -fPIC -c add.c
    - gcc –Wall -fPIC -c sub.c
  - Build the shared library
    - gcc -shared -o libmymath.**so** add.o sub.o
  - To use a shared library, it has to be installed first
    - Copy the library into one of the standard directories (/usr/lib)
    - ldconfig /usr/lib
  - Use the library
    - gcc -o example2 main.o libmymath.so OR
    - gcc -o example2 main.o –lmymath



- Execute command to create static library
  - `ar rs` libmymath.a add.o sub.o
    - -s: write the object files into the archive
    - -r: insert files with replacement if they already exist

Dynamic loading



```
#include <stdio.h>
#include <dlfcn.h>

int main(int argc, char* argv[]) {
  int i = 10;
  void (*myfunc)(int *); void *dl_handle;
  char *error;
  dl_handle = dlopen("libmymath.so", RTLD_LAZY);//RTLD_NOW
  if(!dl_handle) {
    printf("dlopen() error - %s\n", dlerror()); return 1;
  }
  //Calling mul5(&i);
  myfunc = dlsym(dl_handle, "mul5"); error = dlerror();
  if(error != NULL) {
    printf("dlsym mul5 error - %s\n", error); return 1;
  }
  myfunc(&i);
  //Calling add1(&i);
  myfunc = dlsym(dl_handle, "add1"); error = dlerror();
  if(error != NULL) {
    printf("dlsym add1 error - %s\n", error); return 1;
  }
  myfunc(&i);
  printf("i = %d\n", i);
  dlclose(dl_handle);
  return 0;
}
```

- Copy the code into main.c
- gcc main.c -o main -ldl
- You will have to set the environment variable LD_LIBRARY_PATH to include the path that contains libmymath.so

WEEK 9

Communication over the Internet Guarantees

- **Confidentiality**: message secrecy
- **Data Integrity:** message consistency
- **Authentication:** identity confirmation
- **Authorization:** specifying access rights to resources

Encryption Types

- **Symmetric Key Encryption**
  - Shared/Secret Key: key used to encrypt is same used to decrypt
  - Key distribution is a problem, cannot be compromised en route
  - Can be broken with brute force attack, where all possible combinations are generated
- **Asymmetric Key Encryption**
  - Public/Private keys, only creator knows relation between keys
  - Data encrypted with public key can only be decrypted by private key
  - Data encrypted with private key can only be decrypted by public key
  - Public key is publicly accessible
  - Private key is PRIVATE, never goes on any network
- Messages encrypted with public key, only decrypted with private

SSH (**S**ecure **SH**ell) used to remotely access shell

- client ssh's to remote server
- $ ssh username@somehost

Host Validation

- First time talking to server requires host validation, ssh doesn't know host yet
  - Shows hostname, IP address, fingerprint of public key, verify identity
  - After accepting, public key will be saved in ~/.ssh/known_hosts
- Next time connecting, checks hosts/server's public key against saved public key
  - Must match, else is not authenticated
  - Client asks server to prove it is owner of public key with asymmetric encryption
    - Message encrypted with public key, if server is true recipient, then can decrypt message with private key and validate/authenticate

Session Validation

- Client and server use **symmetric** encryption key aka the session key
- All messages encrypted/decrypted with session key
- Anyone without the key will be unable to decrypt the messages

User Authentication

- **Password-based authentication**
  - Prompt for password on remote server
  - Checks username specified exists
  - If remote password is correct, authenticated
- **Key-based authentication**
  - Generate key pair on client
  - Copy public key to server (~/.ssh/authorized_keys)
  - Server authenticates client if it can demonstrate that it has the private key
  - Private key can be protected with passphrase
  - Every time you ssh to host, will be inconveniently asked for passphrase
- **ssh-agent** (passphrase-less ssh)

- Program used with OpenSSH to provide secure way of storing private key
- **ssh-add** prompts user for passphrase once, then adds it to list maintained by ssh-agent. Will not be prompted for it again using SSH
- OpenSSH will talk to local ssh-agent daemon to automatically retrieve private key from it

X Window System

- Windowing system that forms basis for most GUIs on UNIX
- X is network-based system, a program can be run on one computer but displayed on another (X Session Forwarding)

Digital Signature

- Electronic stamp or seal that ensures **data integrity** (data not changed en route)
- Appended or detached
    - detached for compressed files, else becomes unreadably compressed

Generating/Sending a Digital Signature

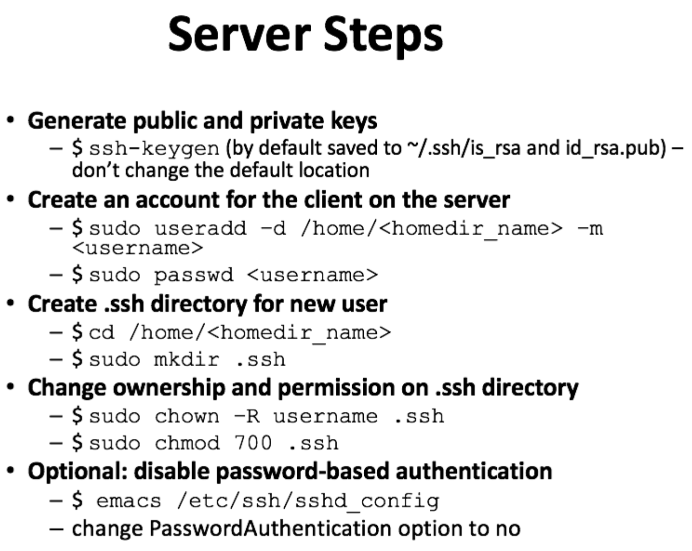
- Generate **Message Digest** through mutually known hashing algorithms
    - Serves as summary of message to transmit; slightest change = different digest
- Creating a Digital Signature
    - Message digest encrypted by sender's private key
    - Encrypted message digest is the digital signature
    - Can only be decrypted with matching public keys
- Append digital signature to message and send to receiver

Verifying/Receiving a Digital Signature

- Recover the Message Digest
    - Decrypt the digital signature using sender's public key, get message digest
- Generate Message Digest from Message
    - Using the same hashing algorithms, generate message digest of received message
- Compare Digests
    - Compare the decrypted message digest of the digital signature, and the one generated from the message
    - If they are not the same, message has been tampered with
    - Digital signature is the one the sender sent since only the sender's public key can decrypt the digital signature
        - That means that decrypting the encrypted message and generating a message digest must be the source of error, aka the message is not what the sender sent
- Digital signature verifies the integrity of the data/message but cannot prove the message's origin.

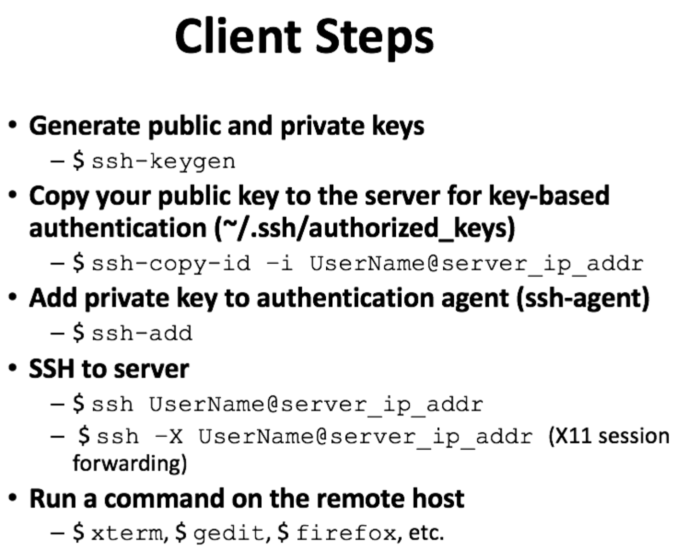
Detached Signature

- Digital signatures can be attached to message or detached
- Detached messages are stored/transmitted separately from message
    - Often to validate software distributed in compressed tar files
    - Cannot sign such a file internally without altering its contents, so signature is detached in separate file
- A Certificate Authority (CA) is a trusted third party that can provide a signature to verify ownership



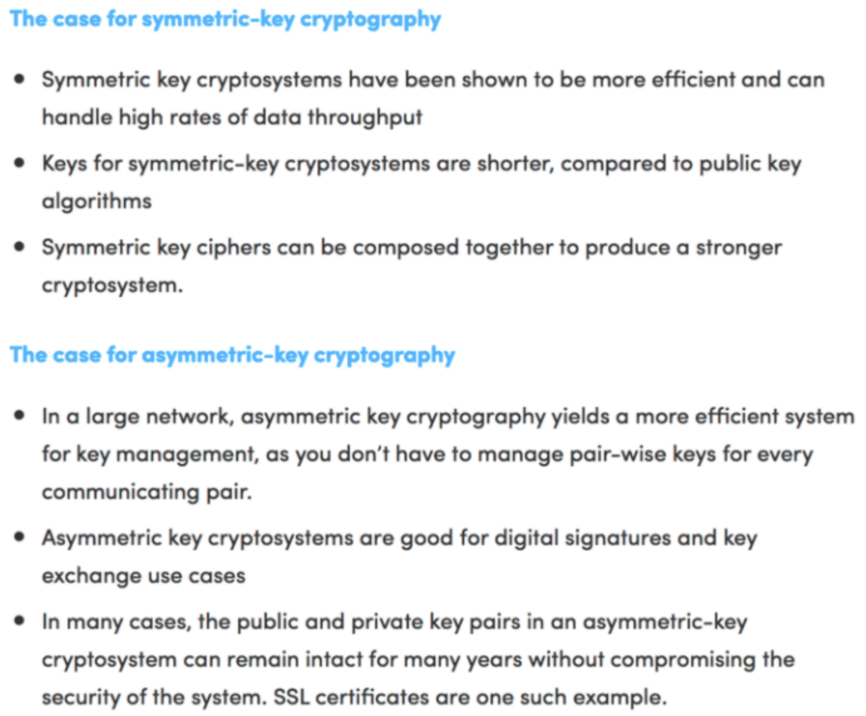
Server Steps

- **Generate public and private keys**
  - $ `ssh-keygen` (by default saved to ~/.ssh/is_rsa and id_rsa.pub) – don't change the default location
- **Create an account for the client on the server**
  - $ `sudo useradd -d /home/<homedir_name> -m <username>`
  - $ `sudo passwd <username>`
- **Create .ssh directory for new user**
  - $ `cd /home/<homedir_name>`
  - $ `sudo mkdir .ssh`
- **Change ownership and permission on .ssh directory**
  - $ `sudo chown -R username .ssh`
  - $ `sudo chmod 700 .ssh`
- **Optional: disable password-based authentication**
  - $ `emacs /etc/ssh/sshd_config`
  - change PasswordAuthentication option to no



Client Steps

- **Generate public and private keys**
  - $ `ssh-keygen`
- **Copy your public key to the server for key-based authentication (~/.ssh/authorized_keys)**
  - $ `ssh-copy-id -i UserName@server_ip_addr`
- **Add private key to authentication agent (ssh-agent)**
  - $ `ssh-add`
- **SSH to server**
  - $ `ssh UserName@server_ip_addr`
  - $ `ssh -X UserName@server_ip_addr` (X11 session forwarding)
- **Run a command on the remote host**
  - $ `xterm`, $ `gedit`, $ `firefox`, etc.

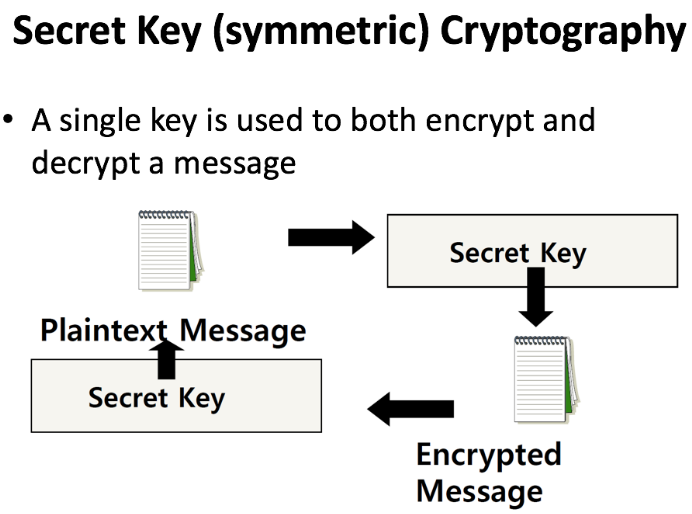


The case for symmetric-key cryptography

- Symmetric key cryptosystems have been shown to be more efficient and can handle high rates of data throughput
- Keys for symmetric-key cryptosystems are shorter, compared to public key algorithms
- Symmetric key ciphers can be composed together to produce a stronger cryptosystem.

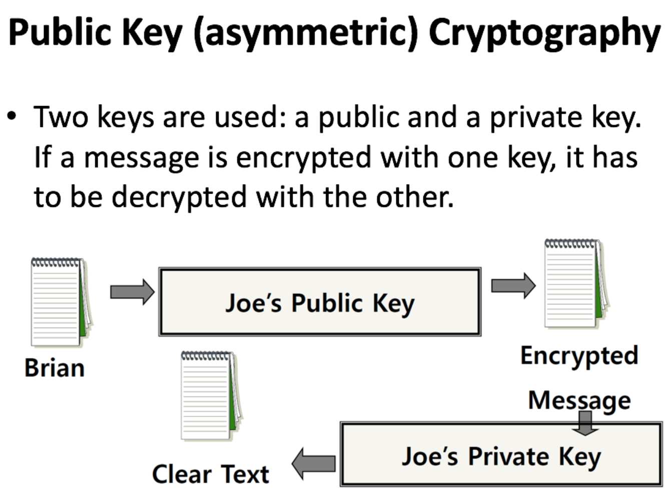
The case for asymmetric-key cryptography

- In a large network, asymmetric key cryptography yields a more efficient system for key management, as you don't have to manage pair-wise keys for every communicating pair.
- Asymmetric key cryptosystems are good for digital signatures and key exchange use cases
- In many cases, the public and private key pairs in an asymmetric-key cryptosystem can remain intact for many years without compromising the security of the system. SSL certificates are one such example.



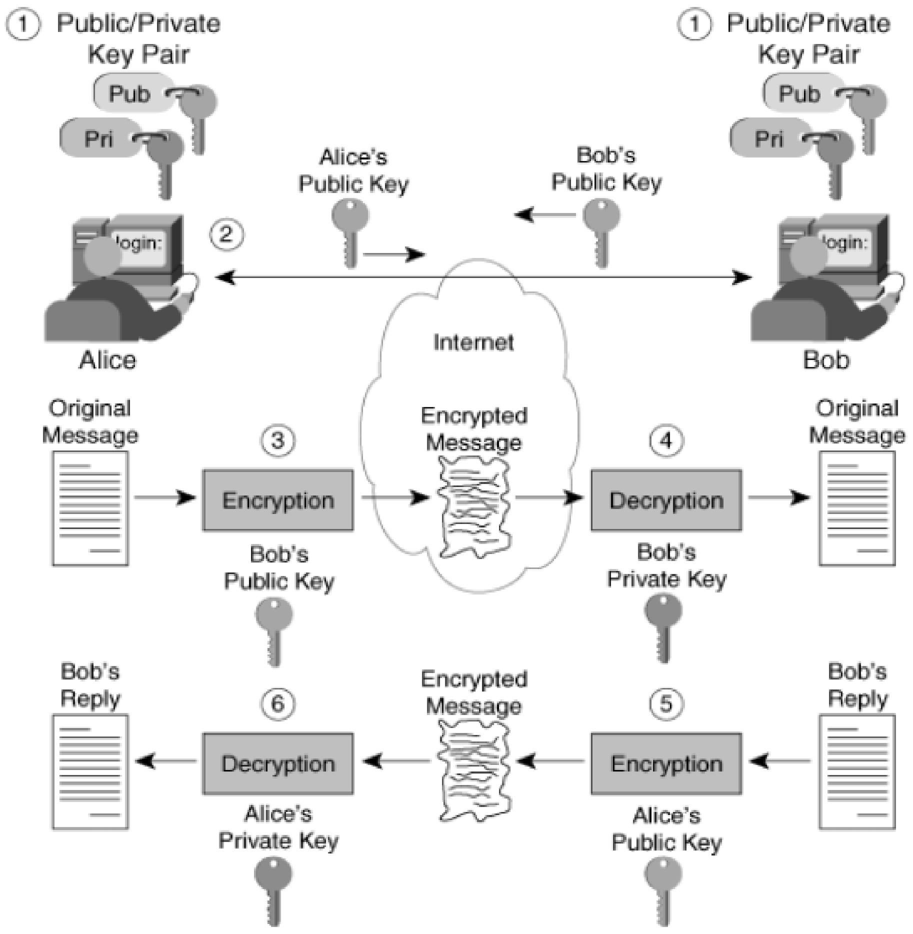
Secret Key (symmetric) Cryptography

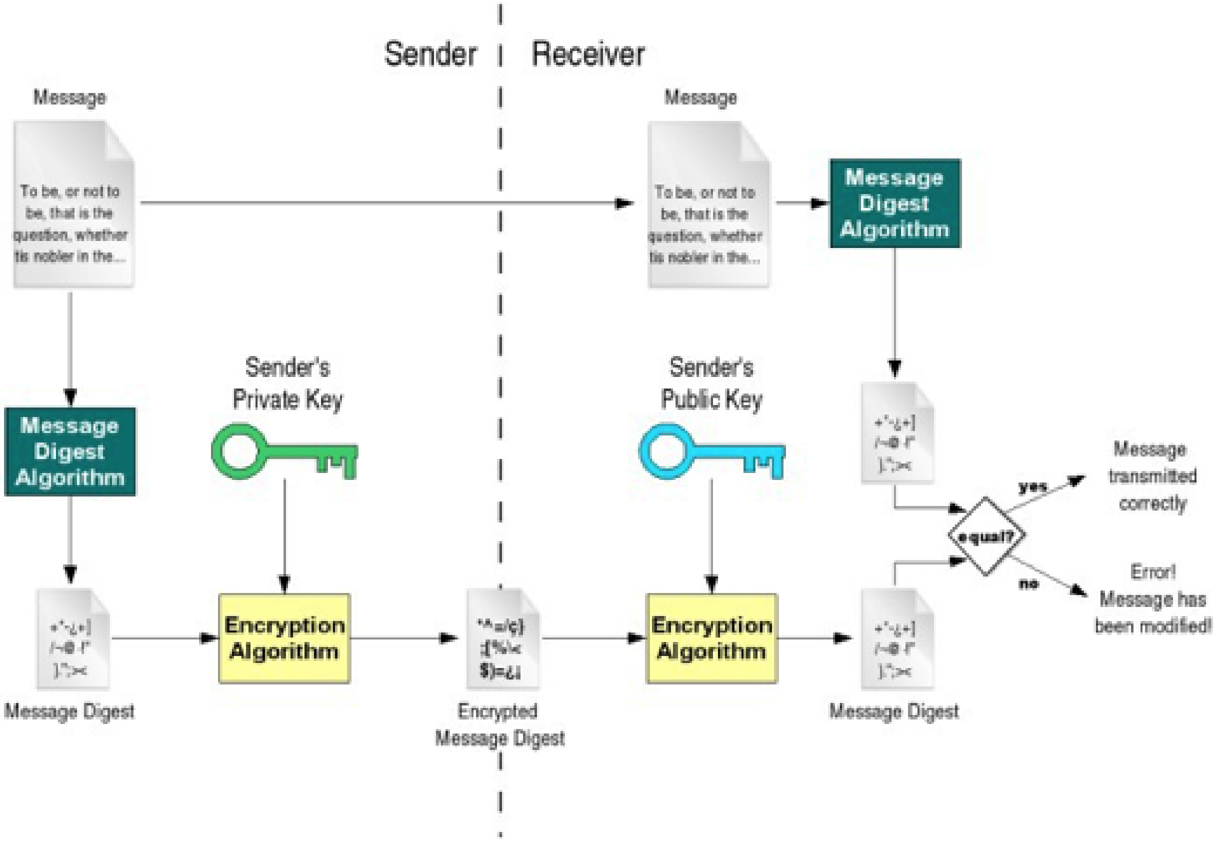
- A single key is used to both encrypt and decrypt a message



Public Key (asymmetric) Cryptography

- Two keys are used: a public and a private key. If a message is encrypted with one key, it has to be decrypted with the other.


1 Public/Private Key Pair
Pub
Pri
1 Public/Private Key Pair
Pub
Pri
Alice's Public Key
Bob's Public Key
2
login:
login:
Internet
Alice
Bob
Original Message
3
Encryption
Encrypted Message
4
Decryption
Original Message
Bob's Public Key
Bob's Private Key
Bob's Reply
6
Decryption
Encrypted Message
5
Encryption
Bob's Reply
Alice's Private Key
Alice's Public Key


Sender
Receiver
Message
To be, or not to be, that is the question, whether tis nobler in the...
Message
To be, or not to be, that is the question, whether tis nobler in the...
Message Digest Algorithm
Message Digest Algorithm
Sender's Private Key
Sender's Public Key
Encryption Algorithm
Encryption Algorithm
Message Digest
Encrypted Message Digest
Message Digest
equal?
yes
no
Message transmitted correctly
Error! Message has been modified!