4-UNIX Tools

grep Command – UNIX

The grep command is used to search text or searches the given file for lines containing a match to the given strings or words.

By default, grep displays the matching lines.
Use grep to search for lines of text that match one or many regular expressions, and outputs only the matching lines.

grep is considered as one of the most useful commands on Linux and Unix-like operating systems.

Simple Syntax: grep “word” filename

Sample Use: grep boo /etc/passwd

chmod Command – UNIX

chmod - change the access permissions
to file system objects like files and directories.

Syntax: chmod *options permissions filename*

Two Examples:

chmod 754 myfile

chmod u-rwx,g-rx,o-r myfile

(The two commands are equivalent.)

diff Command – UNIX

diff - report the difference between 2 text files

compare files line by line

Syntax: diff *options* filenames

Example: diff file1 file2

find Command - UNIX

find search for files in a directory hierarchy

Syntax: find *options* path expressions

Example: find *4*

Would find all the filenames in the current directory that contain the character “4”.

finger Command – UNIX

finger displays information about the system users.

Example: finger –s

-s displays the user's login name, real name, terminal name and write status (the asterisk before terminal name mean that you don’t have write permission with that device), idle time, login time, office location and office phone number.

CSC-60

More about **vi**

Simple vi editing commands

While in Command mode, type:

r replace one character under the cursor
x delete 1 character under the cursor.
2x delete 2 characters (3x, etc.)
u undo the last change to the file

Cutting text in Vi

d^

Deletes from current cursor position to the beginning of the line

d$

Deletes from current cursor position to the end of the line

dw

Deletes from current cursor position to the end of the word

dd

Deletes one line from current cursor position. Specify count to delete many lines.

Cutting & Yanking Text in Vi

While in Command mode, type:

dd	Delete (cut) 1 line from current cursor position
2dd	Delete (cut) 2 lines (***3dd*** to cut 3 lines, etc.)
p	paste lines below current line

Cutting & Yanking Text in Vi

yy yank (copy) a single line

2yy yank (copy) 2 lines (***3yy*** to copy 3 lines, etc.)

P paste lines before current line

Vi Editor

To go to a specific line in the file

:linenumber

Examples (in Command Mode):

1. Go to the 3rd line by typing **:3**
2. Go to the 1st line by typing **:1**
3. Go to the last line by typing **G**

Notes:

(1) :**set number** (to set line numbers)

(2) control-f/b move forward (one page)/backward (one page)

Vi string/search

/[pattern] search forward for the pattern

?[pattern] search backward for the pattern

n search for the next instance of a string

Examples:

1. Search forward for the next line containing the word "printf" by typing **/printf**
2. Search forward for the next instance of **printf** by typing **n**
3. Search backward for the most recent instance of **printf** by typing **?printf**
4. Search backward for the next most recent instance of **printf** by typing **n**

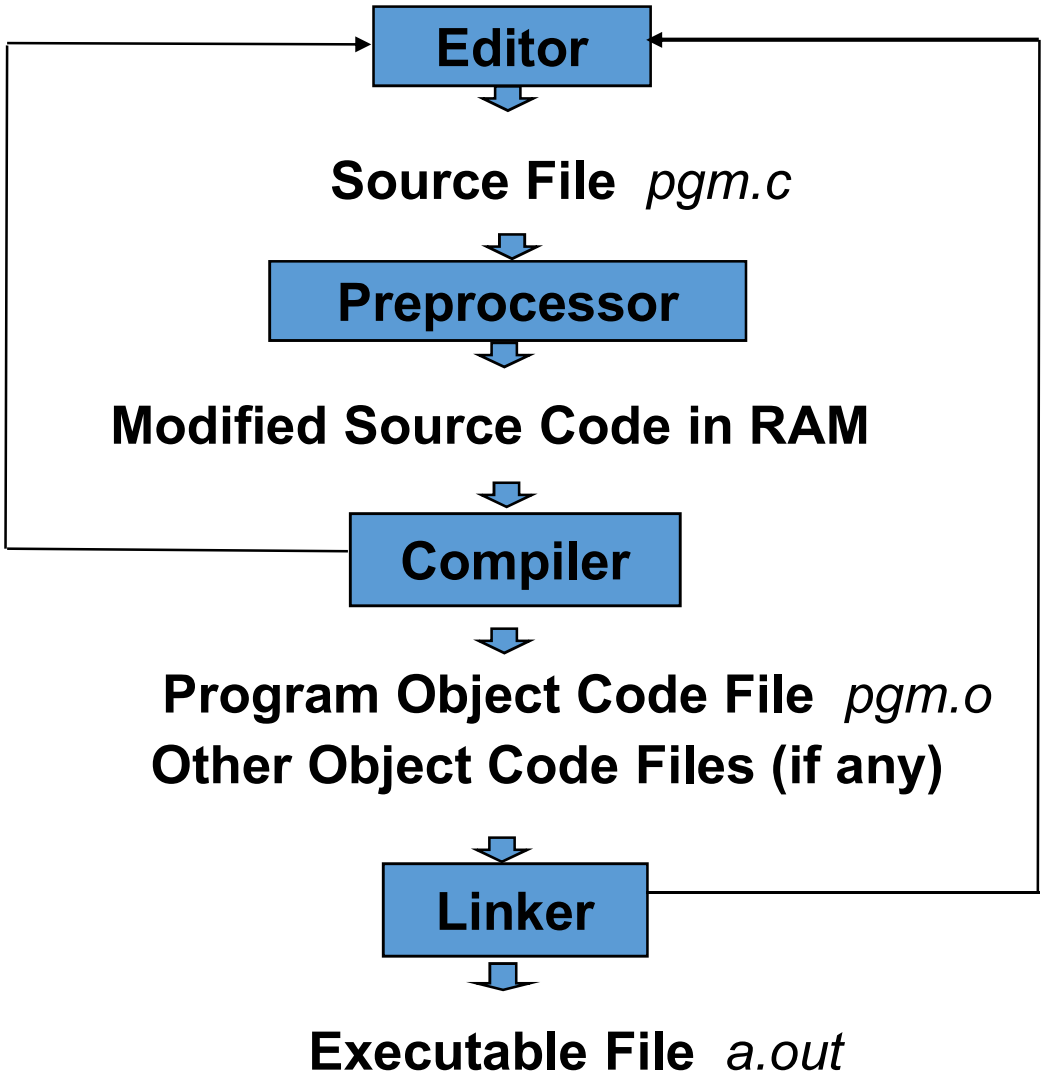
CSC-60
Compilation

What is gcc?

- **gcc** is the GNU Project C compiler
- A command-line program
- **gcc** takes C source files as input
- Outputs an executable: **a.out**
  - You can specify a different output filename

- Note: Although we call this process “compiling a program,” what actually happens is more complicated.

Program Development Using **gcc**



1 of 3 Stages of Compilation

Stage 1: **Preprocessing**

- Performed by a program called the **preprocessor**
- Modifies the source code (in RAM) according to **preprocessor directives (preprocessor commands**) embedded in the source code
- Strips comments and white space from the code
- The source code as stored on disk is not modified.

2 of 3 Stages of Compilation

Stage 2: **Compilation**

- Performed by a program called the **compiler**
- Translates the preprocessor-modified source code into **object code (machine code)**
- Checks for **syntax errors** and **warnings**
- Saves the object code to a disk file, if instructed to do so
- If any compiler errors are received, no object code file will be generated.
  - An object code file will be generated if only warnings, not errors, are received.

3 of 3 Stages of Compilation

Stage 3: **Linking**

- Combines the program object code with other object code to produce the executable file.
- The other object code can come from the **Run-Time Library**, other libraries, or object files that you have created.
- Saves the executable code to a disk file. On the Linux system, that file is called **a.out**.
  - If any linker errors are received, no executable file will be generated.

Gcc example:

“hello.c” - the name of the file with the following contents

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
        printf(“Hello\n“);
         return (EXIT_SUCCESS);
}
```

CSC-60
Debugging

What is gdb?

- **gdb** is the GNU Project debugger
- **gdb** provides some helpful functionality
  - Allows you to stop your program at any given point.
  - You can examine the state of your program when it's stopped.
  - Change things in your program, so you can experiment with correcting the effects of a bug.
- Also a command-line program

Using gdb:

- Compile with the **–g** flag to set up for debugging
- To start gdb with your hello program type:

  gdb HelloProg
- When gdb starts, your program is not actually running.
- Before you do try to run, you should place some break points.
- To start execution, you have to use the ***run*** command. Once you hit a break point, you can examine any variable.

Useful gdb commands

break *place*

place can be the name of a function or a line number

For example: **break main** will stop execution at the first instruction of your program

run *command-line-arguments*

Begin execution of your program with arguments

delete *N*

Removes breakpoints, where *N* is the number of the breakpoint

step

Executes current instruction and stops on the next one

Gdb commands cont.

next

Same as **step** except this doesn't step into functions

print *E*

Prints the value of any variable in your program when you are at a breakpoint, where *E* is the name of the variable you want to print
print/x var (i.e p/x S_IFREG where x is the hex value), other options include: d (decimal), o (octal), t(two - binary), etc.

help *command*

Gives you more information about any command or all if you leave out command

quit

When time to exit gdb

4-UNIX Tools

The End