Lecture 4: CSE 384 (System and Network Programming)

The Linux Command Line, Fifth Internet Edition

Chapters 6-10

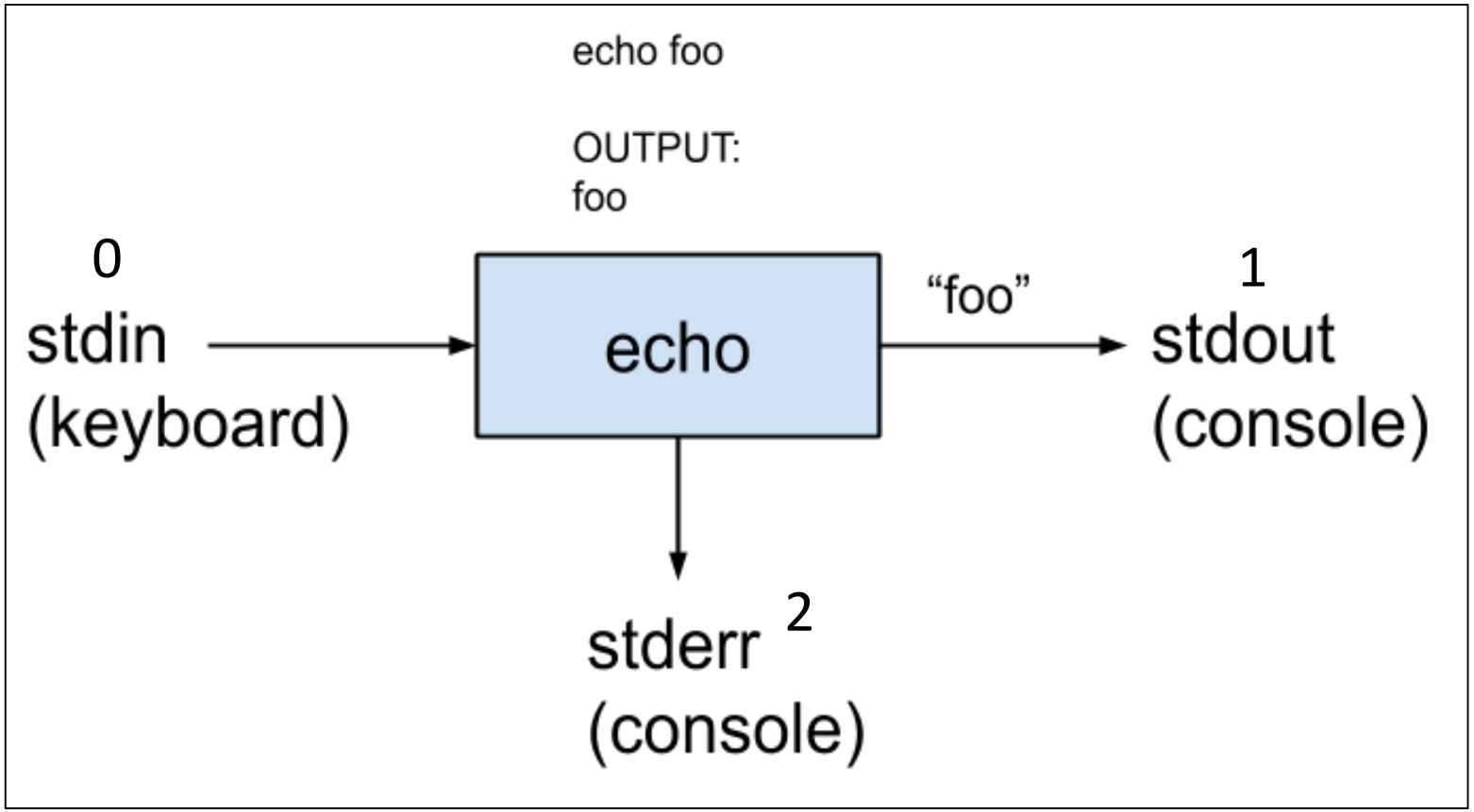
Overview

- I/O Redirection
  - STDIN, STDOUT, STDERR
    - Concept Overview
    - Examples
- Disposing Output
  - /dev/null
- Pipelines
  - Concept Overview
  - Examples

I/O Redirection: based on page 54 of the online version the Linux CommandLine

- redirect the input and output of commands to and from files
- connect multiple commands together into powerful command pipelines
  - In keeping with the Unix (Linux) model, everything is a file….
    - programs (such as ls) send output to a file called standard output: called stdout
    - Status (error) messages to another file called standard error: stderr
      - by default, both stdout/stderr are linked to the screen and not saved into a disk file.
    - Many programs take input from yet another file: *stdin (called standard in)*
      - *By default stdin is connected to the keyboard*
  - *Redirection allows us to change where output (stdout/stderr) goes (e.g. the screen), and where input (stdin) comes from (e.g. the keyboard)*

stdin, stdout, stderr



- In python: “print”
- In Java: “System.out.print”
- In C: “printf’
- fprintf(stderr)
- In C++ “std::cout”
- std::cerr

https://github.com/kennyyu/bootcamp-unix/wiki/stdin,-stdout,-stderr,-and-pipes

```
mwcorley@mwcorley-VirtualBox:~/Desktop$ echo "foo"
foo
```

STDIN, STDOUT, and STDERR are files…

- Operating systems (under the hood), are often a tables with indices that point to resources.
  - Files (and other resources) are often referenced using descriptors (integer numbers) that serve as a handle to the resources.
- STDIN == file descriptor 0
- STDOUT = file descriptor 1
- STDERR = file descriptor 2

```
#define NULL      0
#define EOF       (-1)
#define BUFSIZ    1024
#define OPEN_MAX  20    /* max #files open at once */

typedef struct _iobuf {
    int  cnt;       /* characters left */
    char *ptr;      /* next character position */
    char *base;     /* location of buffer */
    int  flag;      /* mode of file access */
    int  fd;        /* file descriptor */
} FILE;
extern FILE _iob[OPEN_MAX];

#define stdin   (&_iob[0])
#define stdout  (&_iob[1])
#define stderr  (&_iob[2])

enum _flags {
    _READ   = 01,   /* file open for reading */
    _WRITE  = 02,   /* file open for writing */
    _UNBUF  = 04,   /* file is unbuffered */
    _EOF    = 010,  /* EOF has occurred on this file */
    _ERR    = 020   /* error occurred on this file */
};

int _fillbuf(FILE *);
int _flushbuf(int, FILE *);

#define feof(p)     ((p)->flag & _EOF) != 0)
#define ferror(p)   ((p)->flag & _ERR) != 0)
#define fileno(p)   ((p)->fd)

#define getc(p)    (--(p)->cnt >= 0 \
                ? (unsigned char) *(p)->ptr++ : _fillbuf(p))
#define putc(x,p)  (--(p)->cnt >= 0 \
                ? *(p)->ptr++ = (x) : _flushbuf((x),p))

#define getchar()   getc(stdin)
#define putcher(x)  putc((x), stdout)
```

What are files anyway?
Original UNIX "FILE" concept implementation by Ritchie

Source: The C Programming Language 2nd Edition , page 176

Examples

```
mwcorley@mwcorley-VirtualBox:~/Desktop$ ls -l > ls-output
mwcorley@mwcorley-VirtualBox:~/Desktop$ cat ls-output
total 8
drwxrwxr-x 6 mwcorley mwcorley 4096 Jan 22 18:20 code_examples
-rw-rw-r-- 1 mwcorley mwcorley    0 Jan 24 14:29 ls-output
-rw-rw-r-- 1 mwcorley mwcorley    0 Jan 24 14:28 ls-output.txt
```

```
mwcorley@mwcorley-VirtualBox:~/Desktop$ ls -l /bin/usr > ls-output.txt
ls: cannot access '/bin/usr': No such file or directory
mwcorley@mwcorley-VirtualBox:~/Desktop$ cat ls-output
cat: ls-output: No such file or directory
```

```
mwcorley@mwcorley-VirtualBox:~/Desktop$ ls -l /bin/usr 2> ls-output.txt
mwcorley@mwcorley-VirtualBox:~/Desktop$ cat ls-output.txt
ls: cannot access '/bin/usr': No such file or directory
mwcorley@mwcorley-VirtualBox:~/Desktop$
```

Examples

- Redirect STDOUT
  - *ls -l /usr/bin > ls-output.txt*
  - *ls -l /usr/bin 1> ls-output.txt*
- Redirect STDERR
  - [me@linuxbox ~]$ *ls -l /bin/usr > ls-output.txt*
    - What happened?
    - Zero length! The destination file is always rewritten from the beginning.
  - [me@linuxbox ~]$ *ls -l /bin/usr 2> ls-error.txt*
- [me@linuxbox ~]$ *> ls-output.txt*
    - using the redirection operator with no command preceding it will truncate an ex-isting file or create a new, empty file.
- [me@linuxbox ~]$ *ls -l /usr/bin >> ls-output.txt*
- [me@linuxbox ~]$ *ls -l /usr/bin >> ls-output.txt*
  - append redirected output to a file instead of overwriting the file

Examples

- Redirecting Standard Output and Standard Error to One File
  - [me@linuxbox ~]$ ls -l /bin/usr > ls-output.txt 2>&1
    - Using this method, we perform two redirections.
    - First we redirect standard output to the file ls-output.txt
    - then we redirect file descriptor 2 (standard error) to file descriptor 1 (standard output) using the notation 2>&1.
  - [me@linuxbox ~]$ ls -l /bin/usr &> ls-output.txt
    - *Streamlined notation &> in (modern BASH implementations) to re*direct both standard output and standard error to the file
  - append the standard output and standard error streams to a single file
    - [me@linuxbox ~]$ ls -l /bin/usr &>> ls-output.txt

Disposing Output (/dev/null) (page 58)

- Sometimes its appropriate to disregard output from a command, we just want to throw it away.
  - error and status messages.
- [me@linuxbox ~]$ ls -l /bin/usr 2> /dev/null
- Often called the bit bucket
  - Old Unix concept and because of its universality
  - The old Unix joke: If I want to ignore this lecture, you might say that you're sending everything I say to /dev/null

Redirecting STDIN (page 59)

- Consider the Unix cat program – Concatenate Files
  - The *cat* command reads one or more files and copies them to standard output (note wildcard expand is sorted order)

```
mwcorley@mwcorley-VirtualBox:~/Desktop$ echo "Mike" > file1
mwcorley@mwcorley-VirtualBox:~/Desktop$ echo "Corley" > file2
mwcorley@mwcorley-VirtualBox:~/Desktop$ cat file* > file3
mwcorley@mwcorley-VirtualBox:~/Desktop$ cat file3
Mike
Corley
```

  - Now run *cat* with no file arguments (assumes stdin)
    - [me@linuxbox ~]$ cat
      - *the quick brown fox jumped over the lazy dog*.
      - Type Ctrl-d (i.e., hold down the Ctrl key and press “d”)
        - to tell *cat* that it has reached end of file (EOF) on stdin

Redirecting STDIN

- Now run *cat* with no arguments again….
  - [me@linuxbox ~]$ cat
    - Type: *the quick brown fox jumped over the lazy dog*.
    - Type *Ctrl-d* (signal EOF)

```
mwcorley@mwcorley-VirtualBox:~/Desktop$ cat > lazy_dog.txt
the quick fox
mwcorley@mwcorley-VirtualBox:~/Desktop$ cat lazy_dog.txt
the quick fox
mwcorley@mwcorley-VirtualBox:~/Desktop$ cat < lazy_dog.txt
the quick fox
mwcorley@mwcorley-VirtualBox:~/Desktop$
```

- Using the < redirection operator

  the source of stdin changed from the keyboard to the file lazy_dog.txt. We see that the result is the same as passing a single file

Pipelines

- read data from *stdin* and send to *stdout* is utilized by a shell feature called *pipelines*.
- Using the pipe operator | (vertical bar)
  - standard output of one command can be piped into the standard input of another.

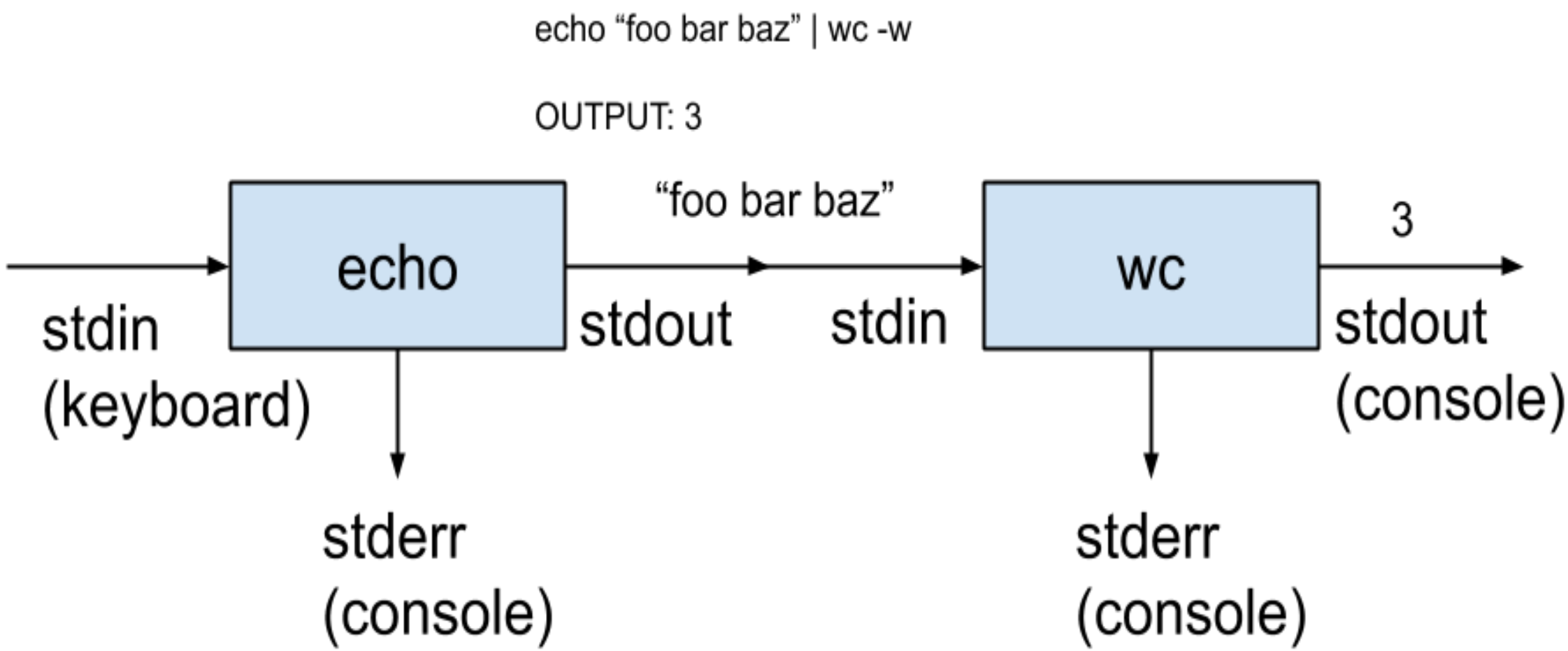
```
command1|command2
```

[me@linuxbox ~]$ ls -l /usr/bin | less

Pipe (|) Versus Redirection (>) - page 62

- The Difference Between **>** and **|**
  - the redirection operator connects a command with a file
    - *command1 > file1 -> echo "hello" > temp*
  - the pipeline operator connects the output of one command with the input of a second command
    - command1 | command2 -> ls -l /usr/bin | less
    - Be carful when you are learning about pipelines
    - What does ***command1 > command2***
      - Answer: sometimes something really bad.
        - # cd /usr/bin
        - # ls > less
          - Notice the # indicates root, (wiped out the less program)

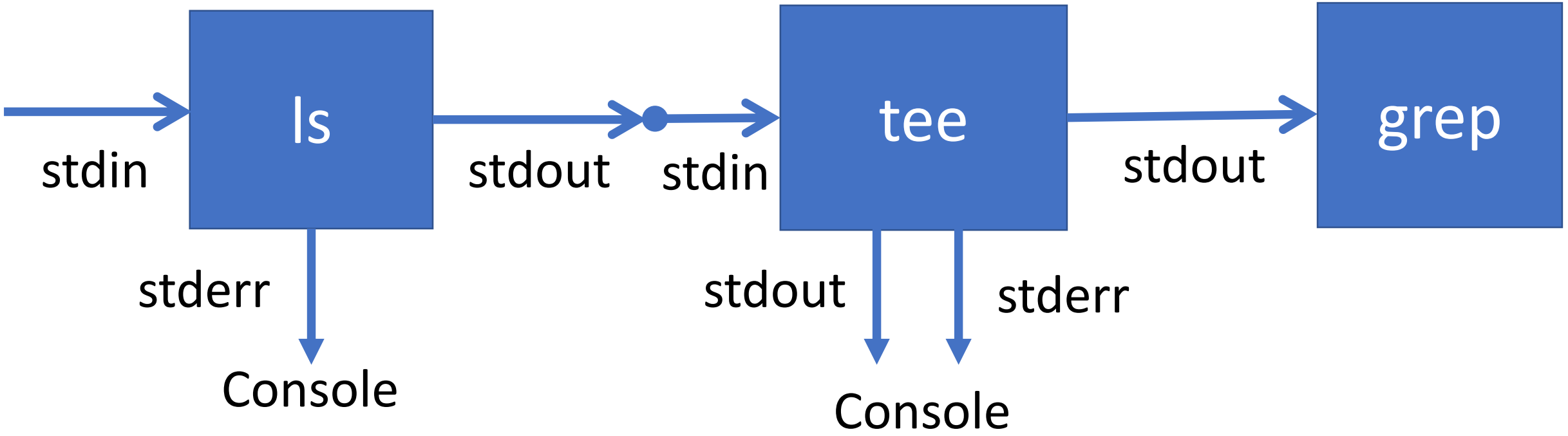
Pipeline (concept)



Source: https://github.com/kennyyu/bootcamp-unix/wiki/stdin,-stdout,-stderr,-and-pipes

Tee command

```
[me@linuxbox ~]$ ls /usr/bin | tee ls.txt | grep zip
```



Filters - piping command together

- https://mwcorley79.github.io/MikeCorley/presentations/TLCL-19.01.pdf#page=86 (The Linux Command Line, William Shotts)
  - sort (page 62) --
  - uniq (unique) -- removes duplicates
  - wc (word count)
  - grep (page 63)
  - head and tail (page 64)
  - tee

Chapter 7: The Shell – Expansion and substitution

- Expansion
  - arithmetic
  - Brace
  - Parameter (shell variables)
- Substitution
- Quoting

- https://mwcorley79.github.io/MikeCorley/presentations/TLCL-19.01.pdf#page=86

Chapter 8: Keyboard shortcuts

- https://mwcorley79.github.io/MikeCorley/presentations/TLCL-19.01.pdf#page=86

- Command history
- Completion
- script

Chapter 9: Permissions

- https://mwcorley79.github.io/MikeCorley/presentations/TLCL-19.01.pdf#page=86
  - id – Display user identity
  - chmod – Change a file's mode
  - umask – Set the default file permissions
  - su – Run a shell as another user
  - sudo – Execute a command as another user
  - chown – Change a file's owner Permissions
  - chgrp – Change a file's group ownership
  - passwd – Change a user's pass

umask

```
mwcorley@mwcorley-VirtualBox:~/Desktop$ umask
0002
```

- controls the default permissions given to a file when it is created.
- octal notation to express a mask of bits to be removed from a file's mode attributes.

chmod examples

- chmod g+rx a.out
  - - give the group read + execute permissions
- chmod g+rwx,o-x a.out
  - - give the group read + write + execute permissions, take way execute for the world
- chmod u+rwx,g+rx,o-rwx a.out == chmod 750
  - Give the owner full access (rwx), the group (rx), take away all access to the world

Special permissions: setuid, setgid, sticky bit

- setuid bit (octal 4000).
  - When applied to an executable file,it sets the effective user ID from that of the real user (the user actually running the program) to that of the program's owner.
    - chmod u+s program

```
mwcorley@mwcorley-VirtualBox:~/Desktop/code_examples$ ls -l /usr/bin/passwd
-rwsr-xr-x 1 root root 59640 Mar 22  2019 /usr/bin/passwd
```

- setgid bit (octal 2000), which, like the setuidbit, changes the effective group ID from the real group ID

- sticky bit (octal 1000)
  - prevents users from deleting or renaming files unless the user is either the owner of the directory, the owner of the file, or the superuser
    - often used to control access to a shared directory, such as /tmp.

chmod examples

chown

SU and SUDO