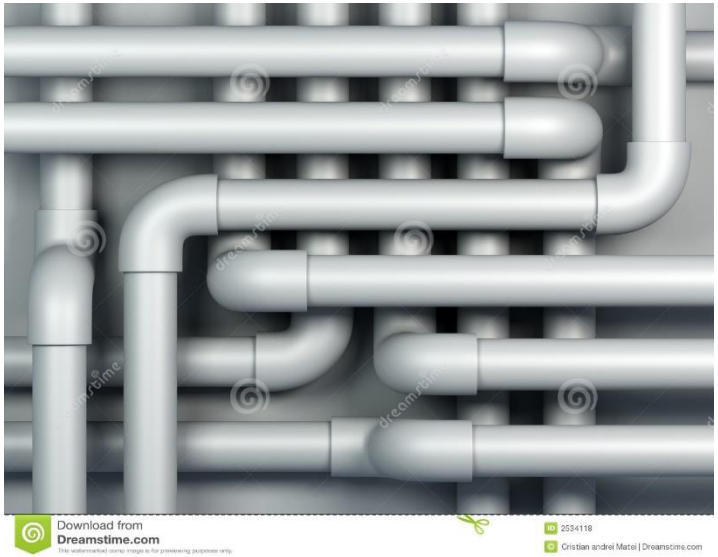
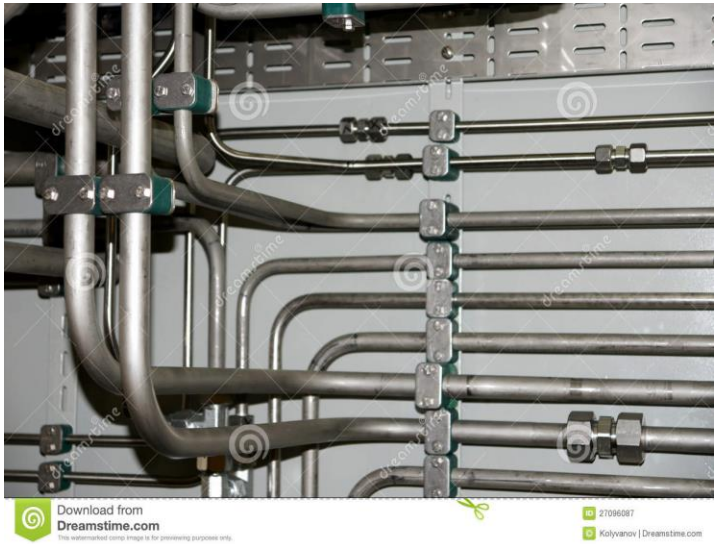
System and Network Administration

Shell, Pipes, Redirection

Shell, Pipes, Redirection





Theory and Practice

- Standard I/O::Redirection
- Standard I/O::Script I/O
- Standard I/O::Netcat
- System Administration::Multitail

- Host Management::Prompt Color
- Standard I/O::ASCII Art

Core Concepts

- A ***process*** is the ***basic unit of activity*** for the operating system
- A ***file*** is an object that a process can ***read from*** and/or ***write to***
  - Every device in the system is represented by a file - even memory is a file.
- The command line is a process known as the ***shell***

stdin, stdout, stderr

- File descriptors are numbers that index a per-process data structure in the kernel that records which I/O channels (file, device, socket, pipe) a process has open. Each I/O system call (read, write, etc.) takes a file descriptor that indicates which channel the call should operate on.

- By convention, the first entry in the table has index 0 and is called standard input (stdin), 1 is the standard output (stdout) and 2 is the standard error (stderr) channel.

Shell, Pipes, Redirection

- When you work from the command line, the shell expects to be getting its input (STDIN) from the keyboard and showing the normal output (STDOUT) and any error messages (STDERR) on the terminal screen.

- The shell allows you to set up connections between these file descriptors and specific files, devices, or pipes to other processes
  - Some of the possible manipulations are rather clever; the implementation that makes it possible is also rather clever.

Pipe

- Perhaps the most commonly used character is | is referred to as "pipe".
- This enables you to pass the output of one command through as the input to another.
- Essentially, the shell ***redirects*** the left side command's **stdout** to the right side command's **stdin**
- This connection applies only to these particular processes, and lasts only as long as the processes run

Pipe

So, for example, if we run the command

```
ls -l | more
```

the output (stdout) of the ls command will be "piped through more". In detail, this command tells the shell to

- 1. create a process for ls
- 2. create a process for more
- 3. connect *ls stdout (FD1)* to *more stdin (FD0)*

Redirect

- The other characters that are used quite often are
- **<** to redirect stdin and
- **>** or **>>** to redirect stdout.

Redirecting stdout to a file is very common, for example the command `ls /bin > myfile`

1. create a process for ls
2. point *ls stdout (FD1)* to a file in the current directory named myfile

> will create a new file every time

>> will add to the end of an existing file, or create a new one

Named Pipe

- Technically, a Unix "named pipe" is a First In First Out (FIFO) interproccess communication mechanism.

- In contrast to "unnamed pipes" (represented by | between two commands) **a named pipe appears as a special file in the filesystem**, which can be accessed by independent processes.

Named Pipe

- Using a named pipe is straightforward, just write something to it in one process and read from it in another.
- Data written to the named pipe will be stored until another process reads from the pipe.
- The first process will exit when it's done writing and sends EOF, and the second process stops when it sees the EOF and exits.

ANSI Escape Codes

- ANSI escape sequences are non-printed text that is interpreted to change the format of some text.
- Their support depends on the terminal, but the color control sequences are commonly supported

```
echo -e "\e[01;32m"; $(date +%R); echo -e "\e[00m"
```

- The -e option of the echo command enables parsing of the escape sequences. Check Quick Reference :: Escape Codes on the menu for a list of possibilities.
- The ; is the way we string together multiple *independent* commands on a single command line

Variable $ubstitution

Echo the STRING(s) to standard output.

- -n do not output the trailing newline
- -e enable interpretation of backslash escapes

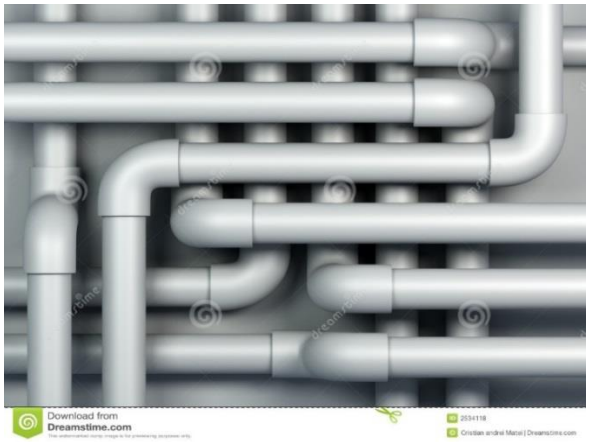
Positional Parameters are provided on the command line

$0 is the name of the command itself

$(command) uses the output of the command
(stdout is saved as a variable

READ is used to parse stdin into variables
(used with pipe)

Shell, Pipes, Redirection



Pipe to a program

```
ls /usr/share/figlet/ | grep .*lf
```

Redirect to a file

```
ls /usr/share/figlet/ > fileToCreate
ls /usr/share/figlet/ >> fileForAppend
```

bash shell

Command substitution

```
figlet $(date +%A)
```

Variable substitution with redirect to a program

```
while read line; do
    printf '%s\n' "$line"
done < "${1:-/dev/stdin}"
```

```
#!/bin/bash

echo "Read from stdin"
read HOST PORT FILE therest

echo -n -e "\e[01:32m"; echo -n $HOST; echo -e "\e[00m"
echo -n -e "\e[01:31m"; echo -n $PORT; echo -e "\e[00m"
echo -n -e "\e[01:34m"; echo -n $FILE; echo -e "\e[00m"
echo "anything else: $therest"
echo
echo "$0 command line positional parameters"
echo -n -e "\e[01:32m"; echo -n $1; echo -e "\e[00m"
echo -n -e "\e[01:31m"; echo -n $2; echo -e "\e[00m"
echo -n -e "\e[01:34m"; echo -n $3; echo -e "\e[00m"
echo

# suggestions:
# echo "$(date +%A) $(date +%x) $(date +%R)" |./ptxt 1 2 3 4
# echo "1 2 3 4" |./ptxt $(date +%A) $(date +%x) $(date +%R)
```

Shell, Pipes, Redirection

```
Send command stdout |to command stdin (pipe)
Send command stdout >to new file
Send command stdout >>to end of existing file (append)

varname="value"
varname="$(command_stdout)"

echo "$varname"     == stdout: value of variable
echo '$varname'     == stdout: $varname (literal string)

SO:
echo '#!/bin/bash' >filename.sh
echo 'fortune |fold -s -w 70 |boxes -d columns' >>filename.sh
      == magic abracadabra is first line of file
      == piped command is second line of file

chmod 755 filename.sh
./filename.sh
```


Q
&
A