Hacettepe University

Department of Computer Engineering

Fall/2011-2012
BIL235 Programming Laboratory I

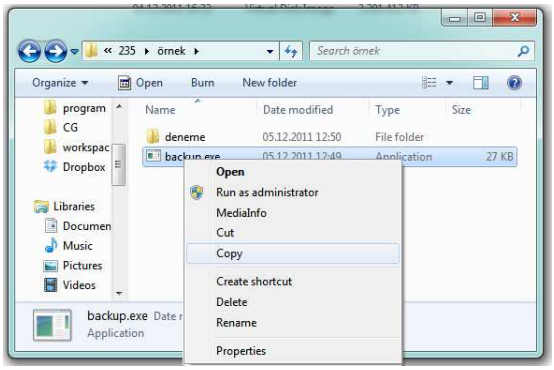
Experiment : 4
Advisors : Dr. Kayhan İMRE, R.A. Cumhur Yiğit ÖZCAN
Subject : Writing UNIX Shell Scripts
Date due : December 23, 2011 - 17.05 p.m.

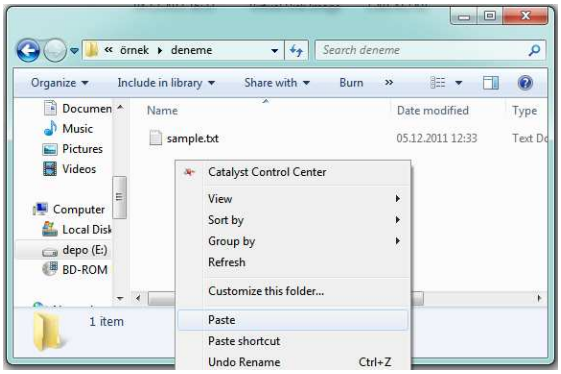
INTRODUCTION AND MOTIVATION

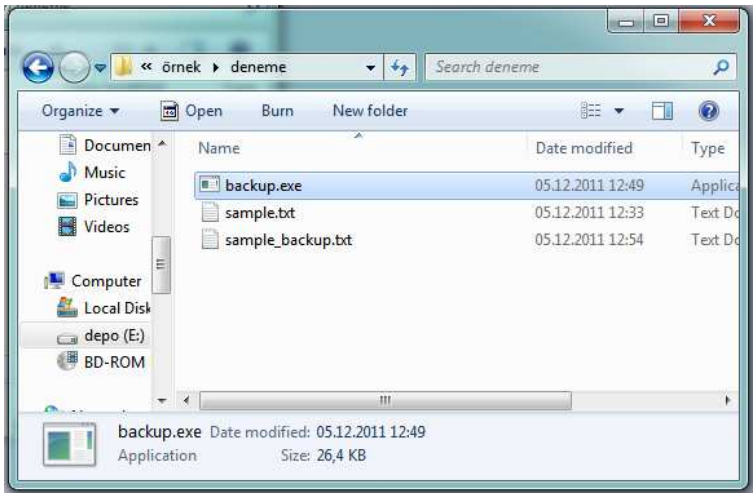
Almost all operating systems provide a **shell** so that users can interact with the system easily. You may think of shell as an interface for OS functionalities. Usually the OS's come with a range of programs, which could be either text based (command prompts) or some kind of graphical user interfaces, to make use of their shell. In the example below, you can see usage of both text based and graphical implementations where user;

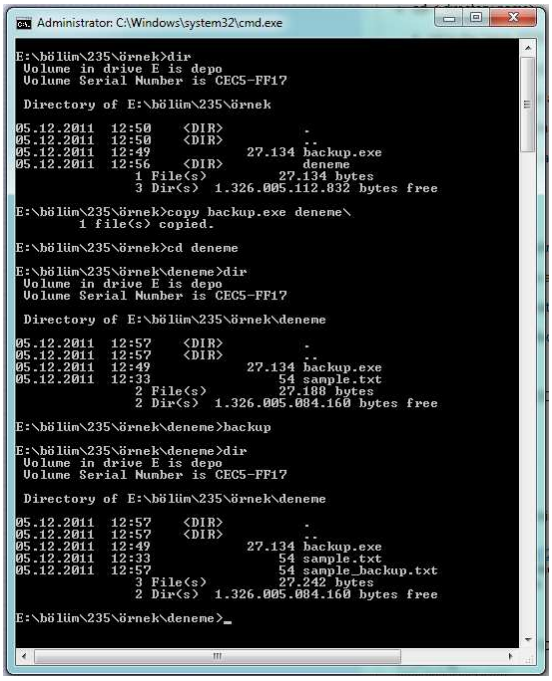
- Copies a backup program to another place
- Changes the current directory where backup program is copied
- Runs the program to backup 'sample.txt'

on Windows 7.









Although a graphical implementation is usually much more useful and easy to use, its capabilities are limited when it comes to automation. Assume that you need to copy and run ‘backup.exe’ in 1000 different locations, everyday. Carrying this task via command prompt or graphical user interface can turn one’s life into a hell.

Solution for this problem comes as ‘shell scripts’, which are –in a way– series of commands packaged in a file. But the reason why shell scripts are very useful and widely popular relies on its capabilities which can be listed as working with files, executing programs, managing I/O redirection, monitoring system state, handling arguments for configuration, converting between different text based formats, using conditional logic, defining and using data structures and doing almost everything you can do with a general purpose programming language and an operating system. Thus, shell script knowledge is a must-have for system administrators, and a very useful skill for every computer engineer.

Scenario

Turkish Government have put a new legislation on, according to which the web domains that contain any inappropriate words will be filtered out and will not be usable by users that runs a filter. The censored word list is published by Information Technologies Institution. Any domain that contains any of these words will be filtered out including all its sub-domains.

FUNCTIONAL REQUIREMENTS OF THE EXPERIMENT

In this experiment, you are a Linux system admin who –as a result of recent events– has to check system users’ files for the censored words. You don't want to miss any of these words in anywhere your server or it will be filtered out completely and your company will lose a lot of money. To carry out this task, you will develop 2 different executable softwares (will be coded in ANSI C) and a shell script.

1. Wordcheck

Independent Role

If you think of *wordcheck* as an independent executable program, it simply takes three arguments from command line and checks whether a file contains any of the censored words and their count. Two of these arguments are paths of files that contains censored word list and file to be checked for censored words. Third argument will determine how the output will be printed (-o for one line output, -f for formatted output). The arguments will be in following order:

Sample usage:

$ wordcheck censored.txt deneme.txt -f censored1 8 censored9 2 censored12 7 $	$ wordcheck censored.txt deneme.txt -o censored1 8 censored9 2 censored12 7 $

Sample censored word list file :

```
censored1
censored2
censored3
    .
    .
    .
Censored20
```

Usage in System

You will use *wordcheck* on files of each user to find out whether they contain any of the censored words and the counts of them. Normally *wordcheck* operates on one file only. Hence, you should run it for each file to be checked, separately.

Notes

- Keep in mind that the third argument is optional and the program should execute as in -f mode if third argument is not given.
- The censored words will contain characters in these ranges : [a..z],[A..Z],[0..9]. Any other character should be treated as a delimiter. Keep this in your mind while parsing the file to be checked. If your parser code exceeds 30 lines, this means you are missing something and wasting your time for nothing.
- You are guaranteed that –in means of this experiment– any word will not be longer than 60 characters. But there are no limits for number of words in a target file or censored word list file.
- Read from censored list file once for every run. You are strongly encouraged to use an appropriate data structure and an algorithm to find if a word is in the censored word list. You will be awarded +5 points as bonus if you explain you data structure and search algorithm in your report (And under right section. It will be discarded if you explain your data structure/algorithm in solution section).

2. Appender

Independent Role

You will use appender to add information collected by *shell script* and *wordcheck* into the report file. Report file will contain information we need after script terminates successfully. An example report file will look like this:

report.txt

```
User : cumhur
Last Login : Sun Dec 4 11:17:57 2011
Word Check Results :
File : /home/cumhur/deneme.txt
censored1 : 8
censored9 : 2
censored12 : 7

File : /home/cumhur/Desktop/folder/deneme2.txt
censored2 : 6
censored4 : 4
censored9 : 37

--------------------
User : oner
Last Login : Sun Dec 4 10:58:33 2011
Word Check Results :

--------------------
User : levent
Last Login : Sat Dec 3 16:54:01 2011
Word Check Results :
File : /home/levent/folder/deneme.txt
censored1 : 8
censored9 : 2
censored12 : 7

--------------------
```

Appender will always take file path of report file (file to append at) as first argument. Second argument could be one of the following:

- -u : Appender will add information of a new user into the report file. This argument will be followed by information about user which are user name and the last login date of the user.
- -c : Appends information about a checked file. This argument will be followed by path of the file which is checked, censored words found in it followed by their count.
- -f : Appender will finalize an entry for a user, simply by putting 20 dash characters at the end of the file.

Sample usage :

```
$ appender report.txt -u cumhur Sat Dec 3 16:54:01 2011
$ appender report.txt -c /home/cumhur/deneme.txt censored1 8 censored9 2 censored12 7
$ appender report.txt -f
```

Usage in System

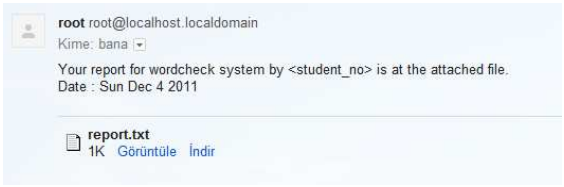
You will use appender to create expected report file. For this purpose, you should run appender once for each user to initialize its entry. Then you should run it for each file checked to add information collected from that file. Finally, you should run it once for each user after you complete checking every file for that user, to finalize that user's entry.

Notes

- Preferably write 3 functions for each functionality of appender and decide which one to call in main function by checking arguments.
- Keep in mind that you should ***append*** to the file.

3. Systemcheck (The Shell Script)

Systemcheck is the shell script that will use wordcheck, appender and some OS commands to create final report about system check for censored words. It will take path of the file which contains list of censored words as its first argument. Second argument will be name of the report file and the third one will be the mail address where the report file will be mailed. Rest of the arguments will be file extensions that will be checked. If no extension is given as command line arguments, script should run wordcheck for each and every file it locates under user's home folder. The script code is responsible for traversing through folders under /home directory to locate files to be checked. It should make proper calls to wordcheck and appender to create expected report file as an output. Finally, it should mail the report file to the mail address given as argument. A copy of the report file should be kept as it is.



Sample usage :

```
$ systemcheck /home/censored.txt /tmp/report.txt cumhuryigitozcan@cs.hacettepe.edu.tr .txt .dat .in
```

Notes

- Examples given for wordcheck and appender are for independent usage. All file path arguments given to the shell script will start from root folder (which is '/' for Linux). Do not try to parse or manage these strings. They will work from wherever they are called.
- Your script should run for every folder under /home directory. Although there are more users in linux systems, you will assume that all the users in system have one home folder under /home directory, their user names are same as these folder names and there are no other users. Don't try to get user list by another way, user list consists of the names you get when you run
    - $ ls /home

  command.
- You will need to *redirect* and/or *pipeline* outputs/inputs of commands and two c programs you develop.

NON-FUNCTIONAL REQUIREMENTS

Requirements for Report

- Follow the conventional report format of the course but particularly pay attention to Software Usage, Problem, Solution, Data Structures, Algorithm, Execution Flow Between Sub-Programs sections.
- Do not copy-paste content from the experiment sheet or *Wikipedia* (or anything that is not your own production).
- ***Please*** write everything under proper sections. Don't explain your bugs under algorithm section.
- Software Usage is for you to explain how this software works, I mean how to operate it properly. This section is like a user manual. Explain the user how he/she can make use of it and what it does in one paragraph. What advisors want to see here is if you have really understood what they wanted you to do, or made up something on your own.
- Problem section is for you to describe the experiment in your own words, not for problems you face during development. Assume that you are explaining what you are supposed to do in means of this experiment to a fellow computer engineer, who has enough knowledge to understand you, from another university without using exactly the same sentences from this sheet. Mention only the key points.
- Solution section is for you to describe how you handled the problem ***briefly***. Again, mention only the key points in a few sentences. We have very beautiful, pretty section called Algorithm for detailed information about your algorithm (even for complexity analysis if you would like to explain). Describing your algorithm in one or two sentences here will be more than enough and I promise that I will read the Algorithm section too.
- Explain important data structures, not every single variable you defined. Valid structures may be arrays of any dimension, linked lists, struct definitions etc. Explain –for example– why did you preferred an array instead of a linked list?
- Give your algorithm in Algorithm section. It shouldn't be either too detailed or too simple. Write as simple (and little) as you can. It will be enough when you think someone who didn't see your code understands what you have done, and will be able to analysis you complexity without any further information.
- For the first time maybe, you have 3 different –so called– programs to develop. Draw a scheme or just explain how they interact in Execution Flow Between Sub-Programs section.
- Don't forget that you should also complete other parts of the report to get full points.

Requirements for Execution Environment

- This experiment is designed so that you cannot run it on dev machine, as you need to get root rights (system admin rights) on the machine it works on. So you ***have to*** install a linux on your own computer. And yes, this is a part of the experiment.
- If you already have a linux system running, you may use it. Any modern *Linux* distribution which has `bash` version 3 or 4 will be more than enough. But you are strongly recommended to download and install CentOS 5 from following links. It is on dev machine so it would be difficult to download it from your home/dorm. Try to download while you are connected to Computer Engineering Department's network directly.
    - http://web.cs.hacettepe.edu.tr/~cumhuryigitozcan/centos/CentOS-5.7-i386-bin-DVD/CentOS-5.7-i386-bin-DVD-1of2.iso
    - http://web.cs.hacettepe.edu.tr/~cumhuryigitozcan/centos/CentOS-5.7-i386-bin-DVD/CentOS-5.7-i386-bin-DVD-2of2.iso

Requirements for Code

- *Place an explanation comment over each command execution line in the shell scripts!*
- Use *meaningful*, English identifiers.
- Indent your script code appropriately. (In general, pay attention to readability of your code.)

Requirements for Submission Format

● You're going submit a zip file named '<student_no>.zip' with the following contents:

```
|-- report
      |-- report.pdf
|-- src
      |-- wordcheck.c
      |-- appender.c
      |-- systemcheck.sh
```

Requirements for Submission Process

- No late submissions are accepted.
- No other submission methods than the online *Submit* system are accepted.

Requirements for Communication

- Use the course news group or Facebook group for communication with your advisors. Keep in mind that the Facebook group is not official and it is there for faster communication (and also if our department's systems are un-reachable temporarily). So you have to check news group often and don't have to check the Facebook group at all. Questions that are asked only via Facebook group will be discarded.
- You can only share your very general *ideas* about the experiment with your friends. Anything beyond that will be considered cheating.
- Read the experiment sheet and inspect the examples located at the FTP site of the course carefully before asking any questions.
- Follow the threads started by your friends in order to catch any issues you may also encounter (and to avoid asking the same question again!).
- If you ask a question via e-mail or seeing me directly, I may need to ask you to copy-paste your question to news group, as otherwise would be injustice for other students. Please be respectful to others.
- Office hours for this course is 13:30 - 17:00 at Thursdays. You don't have to respect these hours but be warned that you may not be welcomed every time. (Office No : 122, next to D9)

Keywords

- Linux, bash, parsing, bash take argument, passing variable as argument, capturing output into variable, redirect output into variable, command pipeline, mail attachment...
- While googling these keywords, using them in combination and adding one of these will make your life easier :
    - how to, tutorial, sample, example...

REFERENCES

- Googling 'Linux bash tutorial', will get you a lot of useful web sites. You don't have to read them all. Just get the idea and start coding. Google what you need when you need it. These three are the ones I could recommend for you to have a quick look at:
    - http://tldp.org/LDP/abs/html/
    - http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html
    - http://www.freeos.com/guides/lsst/

- UNIX/Linux `man` command is your best friend for this experiment and while working with UNIX/Linux in general. Just type
    - **man <command>**

  to get help about any command. If you cannot get an answer, the command may be a *built in* one for `bash`; then you'll need to check `man` page for `bash` itself.
- If you want to learn about *Linux* basics you may check:
    - http://tille.garrels.be/training/tldp/
- If you're serious about `bash` scripting you may dig into:
    - http://tldp.org/LDP/abs/html/index.html