Shell Scripting 2020 third week

Format of the learning diary and scripts

The tasks are returned in moodle.

Return the learning diary in **PDF format**. Preamble the document with the following information:

- Name and student number

The learning diary should contain the answers to questions made in the "Put in your answer" parts. Also example output can be put into the answer.

Include the code in week's directory as a tarball with scripts named with the task name as "task#.sh", i.e. for task 2 the name would be "task2.sh".

Deadline for this set of tasks is Monday 23th of November 2020 and I will release the next set of tasks on 16th of November.

If you run into problems, try to read the man pages and try google. If you are still stuck with the task send me a msg in moodle and I will answer during normal office hours (9-17).

Introduction to variables (17)

There's really only one variable class in Bash and it's a null-terminated string. By using specific keywords, variables can be used as integers in arithmetic. You can list all currently set shell variables with set.

Variables can be further categorized into variable types based on how and when the variables are set.

1. Local variables disappear when the shell session exits: this can be a terminal session or a shell script. Example: set by today=`date +'%Y.%m.%d %H:%m'`, access by echo $today
2. Environment variables set by the shell and/or its parents. Example: echo $PATH, echo $SHELL
3. Special shell variables, typically set by the shell every time a program runs and/or exits. Examples: see below

This categorization is somewhat artificial though, as there is no built-in way of separate the types. Still, it can be useful

Put in your answer:

- Choose three variables from your shell environment and describe what they do.

```
Hint: see section Shell Variables of man bash.
```

Special Shell variables (18)

Since this is a crash course, I can just link the Advanced Bash Scripting Guide's reference cards and expect you to know the special shell variables after that.

```
Hint: It's very handy to keep the reference cards bookmarked in your browser.
```
In practice, it's not so bad. You only need a small subset of the variables in most cases.

- $0 is the name of the currently executing program
- $1 is the first command-line argument (or parameter) passed to the program, $2 is the second, ...
- $? is the return value of the program, set after the program has finished execution.

Put in your answer:

- Write a script called echo that, well, echoes its command-line arguments back to it. For example,

```
$ ./echo.sh
```
returns

```
$
$ ./echo.sh It was a dark and stormy night..
```
returns

```
$ It was a dark and stormy night..
```
and finally:

```
./echo.sh I see a lot of files: `ls`
```
returns

```
I see a lot of files: bash-cat.sh echo.sh ssh-auto-try1.sh today.sh
```

The difference between Bash and Bash (19)

Local variables set within a shell session are not inherited by new shells.

Try it out. Set a variable, then execute a new bash session, and try to access the variable. Once you exit the second session, the variable is available again.

Put in your answer:

- Write a script that illustrates variable visibility. Make your script print out the process identifiers (PIDs) of the shells. This implies the creation of one local variable that would be set in the first shell but would not in a second shell.

`Hint: A bash shell invoked from a bash script will not read the lines following its invocation.`

Remote invocation (20)

Put in your answer:

- Write a bash script that takes two command line arguments as its parameters: a hostname and a command. The script will execute command at hostname, and print out the output returned by the host.

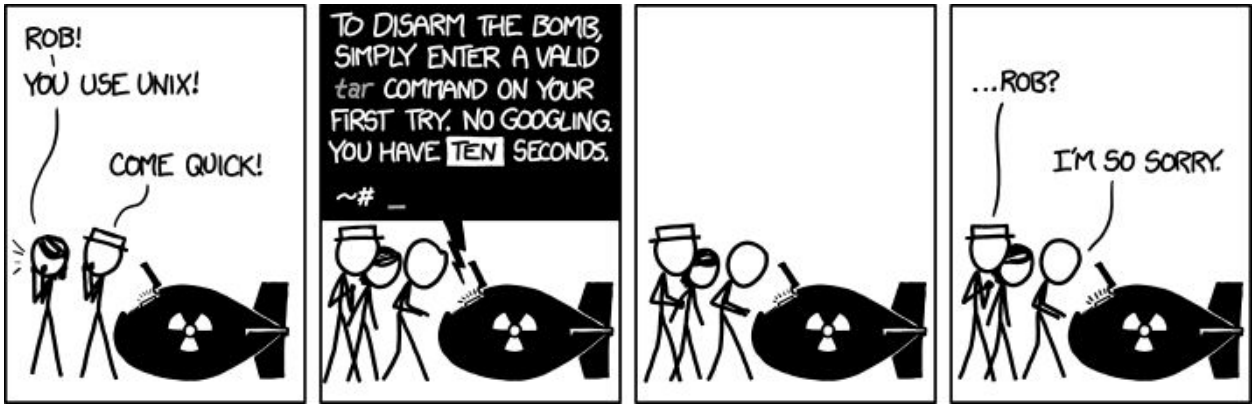
Example:

```
$ ./remote-invocation.sh "melkki.cs.helsinki.fi" "ls"
```
returns the listing of your melkki home folder directory

Tar (21)

The tar command "stores and extracts files from a tape or disk archive." (from man tar). Tape archives are not as useful as they used to be, but disk archives are.

tar's parameters are named a bit differently than many other programs. This is because tar is so old that it precedes some conventions that have later caught on.



Suppose you have a collection of scripts ending in *sh in your subdir. Now, you can create a disk archive that will store the files and their metadata by doing tar cf tarball.tar *sh. The parameter f specifies a file name, while c stands for catenation. List the contents of the file with cat tarball. Notice that the data is still in plaintext format.

Basically any compression program can be used to squeeze the tarball into a fraction of its original space. This is a form of symbiosis: the compression program will understand nothing of files or their metadata, while tar does not need to implement compression algorithms.

Advanced: Two common data compression programs are gzip and bzip2. gzip implements the LZ77 variant of Ziv-Lempel coding, while bzip2 uses the Burrows-Wheeler text transformation algorithm. The course Data Compression Techniques will teach you exactly what their differences are. For now, it suffices to say that bzip2 is more effective, but a little bit more processor-intensive, and maybe a little less widespread.

The general form to combine catenation with compression is tar cf - *sh | compression-program > output-file.suffix. We end up doing this so often that tar has been programmed to provide shortcuts. The file suffixes are chosen by convention. See the tar article on Wikipedia for a full list. For gzip and bzip2 the commands and suffixes are as follows:

- tar cjf destination-archive.tar.bz2 source-files [optional-more-files] creates a tarball and compresses it with bzip2
- tar czf destination-archive.tar.gz source-files [optional-more-files] creates a tarball and compresses it with gzip

Put in your answer:

- Compress your collection of shell scripts (and pictures downloaded last weeks, to get some mass for the files) using (tar with gzip) and (tar with bzip2). Show a comparison of the file sizes from both methods with the compressed tarball.
- Figure out how to bypass tar's shortcut and call bzip2 and gzip explicitly by using a pipeline.

Hints: `tar tutorial`

Local and network file systems (22)

Before we start extracting and working with the following tasks, we must digress shortly to local and network file systems.

During Week 1 exercises, you learned that the Department uses NFS to share or export the same home directories on the workstations.

As there are quite a lot of people on this course, we don't want to overload the NFS system. Therefore, you will be using a local file system for the following tasks. Also, do not use shell.cs.helsinki.fi (alias melkinpaasi) but either a local workstation, a laptop.

Put in your answer:

- Use the command output of mount to figure out which file systems are local. State which host and file system you have decided to use for this week's exercises.

```
Hints: File systems of type ext3, ext4, and tmpfs are usually local.
The contents of a tmpfs consume local memory instead of disk (see for example /tmp
folder).
df -h can be used to list available disk space.
```

Fetch and extract (23)

Fetch and extract

Extractions is very similar, but instead of c for catenate we use x for extract. Before we can do this, we need a compressed tarball to begin with.

Use the file lost24-monitor-temps-and-fans-v2.tar.bz2 for this. You can download it using wget or curl.

https://wiki.helsinki.fi/download/attachments/124126879/lost24-monitor-temps-and-fans-v2.tar.bz2

Put in your answer:

- Uncompress and extract the file in a pipeline, i.e., without first saving the compressed tarball to disk on local file system. Show your command in the report.

Note. we say it once more: do not use your home directory (on file server FS) for the extraction. It will take 1000 times more time than using a local file system.

```
Hint: curl writes to stdout by default, while wget must be explicitly told to do so.
```

Doing your business somewhere else (24)

Sometimes it is useful to use another host for the compression step. Possible reasons might be a more powerful processor or lack of disk space, if the disk archive is very large.

Put in your answer:

- Fetch the tarball from lost24-monitor-temps-and-fans.tar-v2.bz2 to your chosen remote node. Then use ssh so that uncompression happens on that node, but files are extracted to the local file system of the calling host where it is split to files. Present your solution.

```
Hint: Redirect standard output to local tar.
```

GREP and CUT (25)

We will begin by analysing the hp-temps.txt files. We will use grep to select individual lines and cut to separate the line contents into usable variables.

cut is a very simple tool, and I will allow you to become familiar by yourselves. Ask a search engine how to use cut or just read man cut.

Put in your answer:

- Report which unique PROCESSOR_ZONE temperatures (unique temperature) were recorded on 2011.12.25. Your output must return the temperatures in degrees Celsius only. Present your command and its output.

```
Hint: man rev.
Hint: First make your solution work for one file at a time. Then use grep, cut, and
finally sort. Second hint, read through the man pages to find best options for the
job.
```

Don't run with the scissors (26)

cut in this form is fragile, though. Even minute changes in the input format will break cut if the user has specified selections using character counts.

One way to work around this problem is to rewrite the source text in an intermediary phase and then do selection. Next, we will introduce you to sed, which is powerful like a roundhouse kick, but painful to get just right.

sed uses regular expressions extensively, and a good cheat sheet will help you a lot. Unfortunately, I don't know very good ones. You can try these, though:

- http://www.greenend.org.uk/rjk/2002/06/regexp.html
- http://sed.sourceforge.net/grabbag/tutorials/sedfaq.txt section 3.1.1

- http://pubs.opengroup.org/onlinepubs/009695399/basedefs/xbd_chap09.html#tag_09_03
- http://sed.sourceforge.net/sed1line.txt

The problem with sed is that commands quickly become truly illegible. Remember to comment your bash scripts!

Put in your answer:

- Grep PROCESSOR_ZONE like you did in the last exercise and write a sed expression that rewrites consecutive spaces as commas and also rewrites all slashes as commas so your output looks like a CVS file. Present your solution

```
Hint: sed accepts multiple expressions (-e) at once. You can also input the
expressions as a separate text file.
```

Too long to read (27)

Suppose that you would be interested in figuring out what the maximum reported PROCESSOR_ZONE temperature was. For analyzing a single day, the solution might begin with something like grep PROCESSOR_ZONE */hp-temps.txt. And indeed, this would work.

But what if you were looking at all the recorded temperatures, not just for a single day, but for a week, a month, or the whole data set?

That solution would eventually break. And it's not grep's fault. Even something as simple as possible, like ls, will choke on this approach.

Put in your answer:

- Try: ../lost24/monitor/$ ls */*/hp-temps.txt and describe the result in your answer
- Fortunately, we can use the command find Use find to get the location of each hp-temps.txt file from November, 2011. Present the command and sample of it's output.

Advanced: More thoroughly, it is not the shell's fault either. The limits of the virtual memory resources are the root cause for this error type

Escape as a true friend (3.8)

find can handle */*/hp-temps.txt because it doesn't actually rely on shell variable expansion. In fact, shell variable expansion is the enemy of find since they both implement it, but the shell goes first. This is why it is good practice to use 'single quotes' when talking to find.

This detail will probably return to haunt you at some point, so let's be prepared for that occasion. Consider these two forms of calling find:

1. find /path/to/lost24/monitor/2011.12.25 -name *temps.txt
2. find /path/to/lost24/monitor/2011.12.25 -name '*temps.txt'

(The precise subdirectory /path/to/lost24 will vary in your case.)

Put in your answer:

- Construct an example where the first type (1.) fails but the second still works. Present the setup and the output.

We will have a more indepth look into quoting and evaluating strings during the later exercises.

Hint: checkout this example

```
$ touch footest.txt
$ find . -name *test.txt
$ find . -name '*test.txt'
$ touch bartest.txt
$ find . -name *test.txt
$ find . -name '*test.txt'
```

The Immelmann (29)

The Immelmann

As a warmup to loops, let's create some horribly hipster-like photos. Remember the pics we snagged from Exactum-kamera during Week1?

The swiss army knife of image manipulation is called ImageMagick. It has many binary APIs, the least of which is not the shell utility convert. convert features a full salvo of ImageMagick's manipulation filters.

What we shall be doing is to simulate a polaroid-like instant photo application by adding a frame, some rotation, and a bit of sepia to the Exactum-kamera photos. For a single file, the syntax is

Histerify picture

```
inputfile=201007011200.jpg
prefix=${inputfile%.jpg}
outputfile=$prefix-hipstah.jpg
convert -sepia-tone 60% +polaroid $inputfile $outputfile
```
convert works on one file at the time only, so you will need to write a for-loop.

Put in your answer:

- Write a script hipstafy.sh which takes as input a directory name. For each *jpg file, the script will generate a corresponding -hipstah.jpg version using the code above.

Advanced: I separated the prefix out of the input file name using shell substrings. The technique comes from apenwarr's blog post, but be very careful: the latter parts go well above and beyond the contents of this course.

Testing (30)

Testing

There are two main ways of testing for the equality of inequality of variables: The test command and testing operators built into the shell itself. For the latter, see CONDITIONAL EXPRESSIONS in man bash. Fortunately, the syntax is very similar, and not by chance.

When we have two variables which can be evaluated as integers and wish figure out which one is larger, we can do it like this:

Testing

```
prev_value=15
curr_value=25
if [ $prev_value -lt $curr_value ]; then
  echo "Is it me, or is it getting hotter?"
else
   echo "Somebody turn up the heat, I'm freezing..."
fi
```
Be very thorough with the placement of spaces around the square brackets, since the shell interpreter might get confused otherwise.

Hottest day

Instead of using find, iterating through all the PROCESSOR_ZONE temperatures with a loop allows us greater flexibility. We can, for example, keep pointers to the most interesting files.

Put in your answer:

- Write a bash script that finds the file (and temperature) which contains the maximum processor temperature from November, 2011.