Quick Announcements

- IE0 still being graded
- Ignore anything that posted prematurely
- Unfinished business with M2
- Start working on your Case Study

Topic

Case Study Assignment

....

Case Study

Logistics for the Case Study assignment

- Select topic + team of 4 students maximum by October 3rd
- Work with TAs & Alessio to refine the topic
- Prepare slides + 15” presentation (+5” for setup / Q&A)
- Each student is responsible for ¼ of the presentation, slides, and related research
- Present during weeks #13 & #14 (order TBA), attend other students’ presentations
  - Nov 14th / 16th / 21st / 23rd

Case Study

Topic T1 - Alternative shells

- These topics aim for students to apply what they learned with Bash to provide an introduction to a different shell.
- The list of suggested shells includes: Zsh, Fish, Xonsh, nushell.
- The topics to be covered should include:
  - Shell initialization files
  - Shell options and configuration
  - Configuration of auto-complete feature
  - Other substitutions applied to the command line: filenames, processes, arithmetic…
  - Control flow

Case Study

Topics T2 - *NIX Editors

- Some editors in the Linux / UNIX world are legends. The goal of these topics is to explore their extensibility features. Please note that only a part of the presentation should deal with an introduction to the usage. The rest should be focused on the programmatic aspects of these editors (e.g., Emacs is built in and allows extension via a dialect of Common LISP).

- The list of suggested editors includes: Emacs (or XEmacs), Vi (or Vim)

- The topics to be covered should include:
  - Basic and Advanced usage
  - Configuring the various modes
  - Plugin development (demonstrate how to write a very simple plugin in order to illustrate the possibilities in terms of programming the editor to customize it)

Case Study

Topics T3 - Classic *NIX tools & languages

- The list of suggested tools includes: awk, sed

- The topics to be covered should include:
  - Introduction to the syntax and applications (mini tutorial)
  - Demonstration with short scripts
  - Comparison with bash of the above script (if it is even feasible)
  - Comparison with Python (or a high-level language of your choice)

Case Study

Topic T4 - Software Packages Management

- These topics will require students to select TWO different software packages management systems (SPMs) and compare them.

- The list of SPMs includes:
  - Debian's .deb format and the apt and dpkg suite of commands.
  - Red Hat's .rpm format and the yum suite of commands
  - Snap / Appimage / flatpack packages
  - MacOS native packages / homebrew
  - Microsoft winget / chocolatey

- The topics to be covered should include:
  - Command line tools to install, uninstall, and search for packages (mini tutorial)
  - How to package a simple "hello world" program, including some basic dependency

Case Study

Topic T5 – Diff & Patch

- Before git, there were patches. The diff and patch commands are still available and help cast some light on how version control system work.

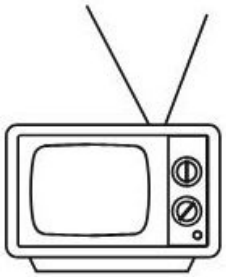
- Commands to be covered: diff, patch, sdiff

- The topics to be covered should include:
  - Using the diff command to generate patches on one file, then folder hierarchy
  - Comparison of the different patch formats
  - Using the patch command to apply patches

Case Study

Topic T6 – LaTeX

- For those who are going to write in an academic setting; e.g., research papers, dissertations…
- Commands to be covered: latex
- The topics to be covered should include:
  - Installation of LaTeX
  - Overview of basic commands, compilation to generate PDF files
  - Available extensions / style templates usage
  - Simple macros and underlying programming



Previously On... CIS4930

- Globbing / wildcards / Glob patterns / filename substitutions
- Bash quoting with \
- Bash globbing options: extglob, dotglob, failglob, nocaseglob, nullglob
- Unfinished business

Topic

Unfinished Business M2T2

Shell Escaping & quoting

Weak Quoting

- Alternative to tedious backslash escaping of a single char at a time
  - Syntax is "...."
  - Turns off MOST of meta-characters substitution… Not all
- Escapes everything but $ \ `
  - `echo "My username is $USERNAME"`
  - `echo "My username is \$USERNAME"`

https://youtu.be/Kb4wdfEOgFo

Weak Quoting and the \ meta-character

\ is not escaped so it is used to escape the following:

\$ \" \\

BUT, \ does not escapes anything else:

```
echo "This is a single \ not escaping anything"
```

→ No need for \\

```
echo "what about \! then \?"
```

→ \ is escaped thus it's not escaping ! or ?
(since these are already quoted)

Strong Quoting

```
echo 'this is *weird* but ok'
```

- Again, less tedious than single \ one meta-char at a time
- Also, escapes $ ` \

Let's explore our limits ☺

- `echo ' Examples: $ ` \ \\ $ \$ " \" that are interesting'`
- `echo ' what about \' then'` → not working since \ doesn't work

How do we fix it?

- ….
- ….

https://youtu.be/N81L0tJ5MT8

Strong Quoting (Solution)

```
echo ‘this is *weird* but ok’
```

- Again, less tedious than single \ one meta-char at a time
- Also, escapes `$` `` ` `` `\`

Let’s explore our limits ☺

- `echo ' Examples: $ ` \ \\ $ \$ " \" that are interesting'`
- `echo ' what about \' then'` → not working since \ doesn’t work

How do we fix it?

- `echo 'what about ' then '` → naïve, does not work, but worth a try ;p
- `echo 'what about' \' 'then '` → we escape OUTSIDE the single quoting

https://youtu.be/N81L0tJ5MT8

Command / Process Substitution

Basic syntax:

- `echo this is the date`
- `echo this is the ` `date` `
- `echo "this is the ` `date` `"` → remember "" leaves ` alone
- `echo 'this is the ` `date` `'` → not working

New, non obsolete, syntax:

- `echo this is the $(date)`
- `echo "this is the $(date)"` → remember "" leaves $ alone
- `echo "this is the $(date -R)"` → options are welcome
- `echo 'this is the $(date)'` → not working

https://youtu.be/HvKfd951GYA

Let's apply this for fun (and profit)*

Useful example

- `gedit myfile`
- `wc -l myfile`
- `echo "there are $(wc -l myfile) lines in myfile"`

Silly example

- `echo $($(echo 'wc -l myfile'))`

Sillier example

- `echo $( echo $(date) )`

Silliest example

- `echo "there are $( echo $(wc -l myfile) | awk '{ printf $1;}' )lines"`

Not-so-silly example

- `echo "The new dirstack is: $(pushd –n $(dirs +4))"`

Topic

M2T3
Bash Environment Variables & Options

Bash has variables

Data types

- Strings (interpreted as integers or floating points numbers sometimes)
- Indexed & associative arrays

Unlike many other programming languages, Bash does not segregate its variables by "type." Essentially, Bash variables are character strings, but, depending on context, Bash permits arithmetic operations and comparisons on variables. The determining factor is whether the value of a variable contains only digits.

-- Advanced Bash Scripting Guide

https://tldp.org/LDP/abs/html/untyped.html

https://youtu.be/WHBKSMKMgK4

How do we use Bash variables?

No need to declare variables

- Assign values
- Interpret them based on context

Basics of using variables:

- `MyVar=42`
- `echo $MyVar`
- `ls $MyVar` ← substitution happens at every command line

Spaces in values: To quote or not to quote?

- `VALUE="myfile"`
- `echo $VALUE` → no "" around it
- `ls -l $VALUE`

- `VALUE="this is my file"` → quote to escape the spaces
- `touch $VALUE` → how many files created?

- `VALUE=myfile` → If no spaces, no need to quote
- `echo $VALUE`
- `ls –l $VALUE`

- `VALUE=\`date\``

Alternative Syntax

```
echo "This is the value that I stored: $VALUE"
```

→ syntax we saw previously

```
echo "This is the value that I stored: ${VALUE}"
```

Allows for the following:

- `VALUE="work"`
- `echo $VALUE`
- `echo "is this $VALUEing?"` → "is this ?" since var doesn't exist
- `echo "is this $VALUE ing?"` → works but I don't want that space
- `echo "is this $VALUE""ing?"` → concatenation, tedious though
- `echo "is this ${VALUE}ing?"` → new syntax

About variables names

Bash is case sensitive for variable names too

- `VALUE=42`
- `echo $value`

Naming Convention

- Upper case for global variables, aka environment variables
  - Environment variables == expected to be used in any program started from shell session
- Lower case for local variables, aka shell variables
  - local variables == vars used on command line (simplification)

Exporting environment variables

`export`

- Set a variable as an environment variable
- Two syntaxes:
  - `export VALUE="something"`
  - `VALUE="something else" ; export VALUE`

Little Experiment

- `echo $VALUE $value`
  - to display both exported and non-exported vars in current shell session
- Start a shell from CLI
- Display the exported, unable to display non exported variables.

→ Subshell has access to environment from original shell

Little Experiment

What if we modify these variables in the inner shell then close it?
Are they modified in the outer shell?

- Define two variables in bash
- Export only one of them
- Start a subshell
- Echo both of the above-variables
- Modify them both
- Exit the subshell
- Echo both of the above-variables