CIS4930 Linux Command Line Interface

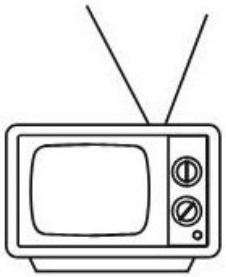
Alessio Gaspar

http://cereal.forest.usf.edu/alessio

Fall 2022

COP3353 Playlist @

https://youtube.com/playlist?list=PLug_bTHghT_lZzSZtIxM6KTbsRyDrpnfP



Previously On... CIS4930

- Syllabus review
- A few word on what's Linux and open source
- More than a few words on Command Line Interfaces

Welcome to Day #2 of the “Surprise Course”

Quick Announcements

Module Section in Canvas

- “slides” posted there
- Reading assignments – to supplement slides
- Remember to take the First Week Quiz!
- Final Exam Matrix

TAs are available

- See URL for Amir (we are working on getting a room)
- Canvas message directly Dan NGuyen until specific hours are set

Module

M01
Basic Usage

Menu for this module

CLI Essentials

- Introduction to the Linux Command Line Interface and its Bash shell.
- We will cover the basics of navigating the file system (more about this topic in the module dedicated to the Linux File System),
- as well as managing processes.

Getting Help

- "Give a man a fish and he is fed for a day, teach him to fish...".
- When it comes to Linux, "learning to fish" boils down to learning to RTFM.
- We are going to learn to use the help tools available in any Linux system.

Topic

What is interacting with Bash all about?

https://youtu.be/Y2gvVXG3f-c

Basic Notions

What is Bash?

- Bourne Shell (sh) → Bourne Again Shell (bash)
- There are alternatives…
  - zsh on MacOS
  - Korn shell
  - C shell
  - TENEX csh
  - Friendly Interactive sh

Where is it used?

- Bash used in Linux / UNIX…
- On windows: Cygwin, git bash…
- On MacOS (used to be Bash, now it's Zsh)

What is the shell interpreting exactly? Commands!

"Commands" == Built-in commands

- The shell is the one interpreting them
- echo
- pwd
  - (The shell's prompt shows the current working directory)
- type
  - type pwd
  - type type
  - type date
  - type -a date

Shell	Full Shell Name	# of built-in commands
sh	Bourne Shell	18
ksh	Korn Shell	47
csh	C-Shell	55
bash	Bourne Again Shell	69
tcsh	TENEX C-Shell	87
	FreeBSD Shell	97
zsh	Z Shell	129

“Commands” == Keywords: if while for …

- Also interpreted by the shell itself

“Commands” == Functions: skip until we script

- You guessed it; also interpreted by the shell itself

“Commands” == External Commands

- date
- type date
- type -a date
- Let’s go meta → bash or sh

Topic

Bash Aliases

https://youtu.be/vIGK7A3i6_Q

Let us look at a few aliases

- Look at the alert alias
  - aliases == 1-line scripts
- Look at the ls alias
  - Commands have options
    - One letter (BSD Style)
    - One letter (standard) –a –b –c
    - Full word --help --version
  - Examples
    - ls –l
    - ls –a → Hiding files with dot
- Order does not matter & we can put them together
  - Ls –a –l → ls –l –a → ls –al → ls –la
  - → order does not matter when these are toggles
- Bash is case sensitive
  - ls –a
  - ls –A
- Full word options
  - ls --help –version

Defining our own Aliases

- Aliases can be removed
  - unalias ls
  - Test it to show lack of color!
- They can be defined
  - alias ls='ls --color=auto'
- They go away when closing the shell
  - Close shell
  - Reopen it
  - Check for alias presence
  - → see bash config files later
- What can we (re)define aliases on?
  - → external commands
  - → builtins

    alias type='type -a'

View aliases as pre-processing of the command line string before we determine whether we are going to execute a builtin or an external command.

Side remark: use single quotes for now

alias stuff=‘echo ‘ ‘	The single quote is interpreted as closing the first single quote	>
alias stuff=$‘echo \‘’	$’ is a special notation allowing ANSI escaping inside the single quoted string Escapes the \’ to ’ Alias tries to echo ‘ but this is an unclosed string so PS2 appears	stuff >
alias stuff=‘echo ‘”’”	Concatenating a single quote in double quotes The single quote is added, same problem than above	stuff >
alias stuff=’echo’ “\’”	Concatenating an escape single quote in double quote The alias try to do echo \’ This results in displaying the actual single quote	stuff ‘
alias stuff=“echo \’”	Using double quotes instead works	stuff ‘
alias stuff=“echo ‘”	Again, this would result in the alias expanding to echo ‘ which lacks a closing single quote	stuff >

Quick Announcements

Register on Piazza

- Demo on Canvas
- All announcement about the course will be there instead of in traditional Canvas announcements

Supplements to address lack of textbook

- Reading assignments in Canvas
- URL of web resources directly in the slides to expand on them
- URL to videos developed for the online / IT version of this course (when appropriate)

Quick Announcements

GQ01 in the queue to release

- Check availability and due date in Canvas syllabus section
- Timed, proctored, 1 attempt only
- Syllabus says textbook is allowed but for this semester this means closed book
- Personal work only

Slides

- Posted before lecture
- Reposted after
- Cumulative slide deck w/ corrections applied to previous sections
- Filename has date + number to help you know if something on canvas is newer than what you already downloaded

For instance, everything past this was added at the last minute and is therefore not in the Canvas version yet

“Quick” Announcements

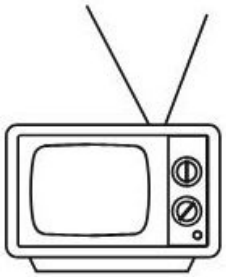
Working on PA + Q&A sessions w/ TAs

- Format or schedule not yet finalized
- Bring your laptops to all sessions just to be safe
- These will also serve as review session / preparation for the upcoming IEs

CSE VMPortal

- All accounts should be opened now
- Access it at http://vmportal.cse.usf.edu/
- Make sure to be on campus or on VPN when you do

- UTM on M1
  - Looks like an option, downloaded it, did not test it yet



Previously On... CIS4930

- Types of Commands interpreted by the shell
  - Builtins / keywords / functions / external commands
- Syntax for 1-letter and full-word options
- Pre-processing applied to your command string
  - aliases
  - It can get more complicated: alias stuff="echo this ain't easy"

Alias w/ multiple commands in one line

- Semicolon
  - date ; echo “is a nice day”
  - date ; echo “The date command worked”

- What if it didn’t work?
  - date -meow ; echo “The date command worked”
  - → we want to insert a conditional here…

https://youtu.be/vIGK7A3i6_Q

Conditional Sequential Execution

&&

- Review on shortcut evaluation of Boolean expressions in C, Java, Perl…
- Application to shell:
  - date && echo “it worked”
  - date --meow && echo “it worked”

||

- date –meow || echo “it did not work”

How does it know?

- Concept of exit status
- echo $?

Executing commands in subshells

Setup

- TAG="I am the original"
- echo $TAG

Group commands in Subshell

- (echo $TAG)
- (TAG="not sure anymore" ; echo $TAG)
- echo $TAG

Group commands in current shell

- { echo $TAG ; }
- { TAG="what about now“ ; echo $ TAG }
- echo $TAG

Topic

FileSystem Concepts

https://youtu.be/j1l_C0b1ZK4

Absolute vs relative pathnames

- pwd → path name
- Root == origin
  - Not using letter drives like in windows (multiple origins)
  - In Linux / == One root to rule them all
  - Where are all the disks / partitions?
    - mount

- absolute path
  - Refers to folders → ls /home → ls –al /home
  - Refers to files

- Relative path
  - Relative to what? → CWD
  - ls filehere
  - ls folder/filethere
  - . and .. Special folders / notations

- Examples of using pathnames with ls
  - ls .
  - ls ..
  - ls ../tux/
  - ls /home/tux/../../home/././././tux

- Some handy utilities
  - basename /home/tux/myfile.txt
  - dirname /home/tux/myfile.txt

Topic

Moving around in the Linux filesystem

https://youtu.be/k3hokNCQwPw

Moving around w/ builtin commands

- Pwd → Uses ~ abbreviation when applicable

- cd absolute_pathname
- cd relative_pathname

- cd /home/darthtux/

- cd ~

- cd –

Concept of Directory Stack

- type dirs pushd popd
  - Also built-in commands

- Viewing the stack
  - dirs → note that the first entry is always the CWD
  - dirs -l
  - dirs -p
  - dirs -p -l → note that -pl does not work here (builtin)

- Adding to the stack
  - pushd -n /some/where
  - pushd /some/where
  - dirs –p

- Removing from the stack
  - dirs -p
  - popd +0 -n
  - …

Directory Stack access

- dirs –v

- dirs +0
- dirs +1

- dirs -0
- dirs -1

Topic

Creating & removing files & folders

https://youtu.be/xoOAsbnjHJw

Let's start with files

- touch existingFile.txt
  - → modifies date
  - Check it with → `ls -l`
- touch newFile.txt
  - → creates empty file
  - Check it with → `ls -l`
- Removing existing file
  - rm existingFile.txt

What about creating folders?

- Creating empty folder
  - mkdir something
  - mkdir something/else

- mkdir COP3353/m01 → FAILS
  - mkdir --help
  - Check out the –p option

```
mkdir -p COP3353/m01
mkdir -p COP3353/m02/slides/23/
```

- Visualizing hierarchy
  - tree
  - tree -d

What about deleting folders?

Setup

- mkdir COP3353/m01

rmdir COP3353 → warning non empty folder

rm –rf COP3353 → will work

WATCH OUT!!!!

- Uis use TRASHES, CLI does not → watch out
- Not a safe delete either though → forensic / shredding

Topic

Copying, moving & renaming files & folders

https://youtu.be/wc6yR8dbby4

Copy a single file

- cp COP3353/readme.md backup
- Absolute + relative pathnames usable!

Rename one file on the fly

- cp COP3353/readme.md backup/cop3353.md

Copying multiple folders or files

- cp COP3353 COP2512 COP2513 ./backup/
  - → omitting folders
  - cp –r COP3353 COP2512 COP2513 ./backup/
  - → multiple sources and one destination → we cannot rename just copy

What about renaming?

- Use 1 source + 1 destination only
- cp –r COP3353 ./backup/CIS4930

mv

mv something something-else CEN6084/

- Multiple sources + 1 destination

mv something COP4610/something-entirely-different

- 1 source + 1 destination → allows renaming on the fly

Renaming without moving

- mv COP3353 CIS4930

Topic

Bash processes management

- So far
  - Execute 1 thing at a time
  - Wait for it to complete
  - Enter the next command

- Let's simulate a process that runs longer than instantly
  - type sleep
  - sleep 4

- If command takes a while, you might want to still be using your shell
  - → execute the command in the BACKGROUND
  - As opposed to FOREGROUND

- sleep 5 &
  - Gives us a job ID then a PID
  - Job is specific to the bash where you started → show 2 shells side by side
  - PID is global to the system

https://youtu.be/6N7bNvKCJtM

Long-duration processes should be launched in BG

- xeyes → bad
- xeyes & → oh yeah, sweet sweet 90's tek

- Actually, gedit does this automatically for us

Controlling foreground processes w/ shortcuts

- ^C → terminates the process
- ^Z → freezes the process instead of terminating it

Example

- xeyes something ^Z
- sleep 30 &
- jobs
- fg %1 → put back in foreground
- ^Z → ok we froze it again
- bg %1 → ok but how to put it in background?

https://youtu.be/6N7bNvKCJtM

Monitoring Processes / PIDs

Basic syntax:

- ps -e
- ps -f
- ps -eo pid,ppid,ni,comm

Tree views

- ps -ef --forest
- pstree

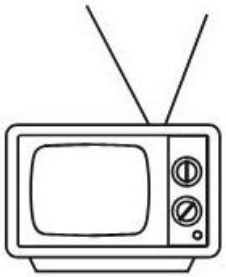
Live Monitoring

- top
  - forest mode available too: V to enable
  - v to toggle displaying children for a given process
- htop
  - Use F5 for forest view

Quick Announcements

GQ-01 & UTM

- See post on Piazza



Previously On... CIS4930

- Basics of CLI filesystem navigation
  - pwd cd mkdir rmdir rm
  - dirs popd pushd
- Basics of CLI process management
  - ^C ^Z
  - fg bg jobs
- Coming up next…
  - More about CLI process management

Prioritizing Processes

- Niceness of a process
  - From -20 to 19, default is 0
  - -20 is highest priority
- Priority of a process
  - PRI = 20 + N in [0:39] or [100:139] for kernel ([1:99] is for real-time)
- Start a process w/ niceness value != 0
  - `nice -n 5 run_my_backups.sh`
  - `nice -n -20 do_this_right_now.sh`
- Renice to change it, later, dynamically
  - `renice -n -20 –p 70899`
  - Try to renice at -20 ☺

https://www.tecmint.com/set-linux-process-priority-using-nice-and-renice-commands/

Topic

Sending signals to processes

https://youtu.be/vSLvhQtAGV4

Sending signals to job IDs or PIDs

- kill → Misunderstood command; it does not KILL processes, but sends them SIGNALS
- kill %1 → ok this one kills the process, it's the default
- kill 8834 → works with PID too

Show me the signals!

kill -l

Signal	Value	Action	Comment
SIGHUP	1	Term	Hangup detected on controlling terminal or death of controlling process
SIGINT	2	Term	Interrupt from keyboard
SIGQUIT	3	Core	Quit from keyboard
SIGILL	4	Core	Illegal Instruction
SIGABRT	6	Core	Abort signal from abort(3)
SIGFPE	8	Core	Floating point exception
SIGKILL	9	Term	Kill signal
SIGSEGV	11	Core	Invalid memory reference
SIGPIPE	13	Term	Broken pipe: write to pipe with no readers
SIGALRM	14	Term	Timer signal from alarm(2)
SIGTERM	15	Term	Termination signal
SIGUSR1	30,10,16	Term	User-defined signal 1
SIGUSR2	31,12,17	Term	User-defined signal 2
SIGCHLD	20,17,18	Ign	Child stopped or terminated
SIGCONT	19,18,25	Cont	Continue if stopped
SIGSTOP	17,19,23	Stop	Stop process
SIGTSTP	18,20,24	Stop	Stop typed at terminal
SIGTTIN	21,21,26	Stop	Terminal input for background process
SIGTTOU	22,22,27	Stop	Terminal output for background process

The signals **SIGKILL** and **SIGSTOP** cannot be caught, blocked, or ignored.

Signal	Syntaxes	Notes

SIGTERM	`kill 9078`
	`kill -TERM 9078`
	`kill -s TERM 9078`
	`kill -15 9078`
		→sends SIGTERM #15 by default
		asks politely for process to terminate
		(process may refuse)

SIGKILL	`kill -KILL 9078`
	`kill –s KILL 9078`
	`kill -9 9078`
		→sends SIGKILL #9 explicitly
		Does not really "ask" but just shuts down the process

SIGSTOP	`kill –STOP 9078`
	`kill –s STOP 9078`
	`kill -19 9078`
		→ same as `^Z`

SIGCONT	`kill -CONT 9078`
	`kill -s CONT 9078`
	`kill -18 9078`
		→ same as `fg` / `bg`

Topic

Processes Lineages & Signals Propagation

A simple Experiment…

- Launchme.exe &
- jobs
- [kill the terminal window]
- ps -ef |grep launchme.exe [in another terminal]

[note: if we just exit-ed the shell, it would not send SIGHUP to its bg jobs]

Run process in foreground (started from an interactive shell, connected to a terminal

So let's assume you've just typed foo:

https://en.wikipedia.org/wiki/SIGHUP

- fork → The process running foo is created.
- The process inherits stdin, stdout, and stderr from the shell.
  - Therefore, it is also connected to the same terminal.
- If the shell receives a SIGHUP, it also sends a SIGHUP to the process
  - (which normally causes the process to terminate).
- Otherwise, the shell waits (is blocked) until the process terminates or gets stopped.

Celtschk @ https://unix.stackexchange.com/questions/3886/difference-between-nohup-disown-and

Run process in background with &

- The process running foo is created.
- It inherits stdout/stderr from the shell (so it still writes to the terminal).
- It also inherits stdin, but as soon as it tries to read from stdin, it is halted.
- It is put into the list of background jobs the shell manages, which means:
  - It is listed with jobs and can be accessed using %n
  - It can be turned into a foreground job using fg
  - If the shell received a SIGHUP, it also sends a SIGHUP to the process

Celtschk @ https://unix.stackexchange.com/questions/3886/difference-between-nohup-disown-and

Disown removes the job from shell's job list

- Disown → all the subpoints above don't apply any more
  - including process being sent SIGHUP by shell.

- However, it still is connected to the terminal
  - so terminal destroyed → program fails if tries to access stdin or stdout.

Conclusion?

- No proper handling for tasks meant to run on a server after logout

Celtschk @ https://unix.stackexchange.com/questions/3886/difference-between-nohup-disown-and

nohup separates process from the terminal:

- It closes stdin
- It redirects stdout and stderr to file nohup.out
- It prevents the process from receiving a SIGHUP (thus the name).
- Does not remove the process from the shell's job control
- Does not put it in the background

Conclusion?

- Ideal for running bg processes while logged out of a server

Celtschk @ https://unix.stackexchange.com/questions/3886/difference-between-nohup-disown-and

Topic

Manpages & The Online Manual

Before we start: The less pager

- `less something.txt`
  - Up / down / pg up / pg down / enter / space
  - h → help page
  - q → quit
- Searching with / and ?
- Running a shell command with !date
  - ! is referred to BANG; BANG command

"Less & more are more or less the same thing, but less is more."
-- Anonymous

https://youtu.be/hM42QDeO7Ic

How to get help about a command?

Just type wrong command

- `mkdir`
- → displays error or help message

Ask for command's help message (usage)

- `ls --help`

help

- → tells you about all builtin commands
- `help type`

Man

- Check out its manpage

https://youtu.be/W-keag_QSOo

Accessing a Command's Manpage

- The "online" manual
  - Online as "right here"
  - Not online as on the web

- `man ls`
  - `Less` pager is used to display the page
  - All shortcuts are available
  - Q when done

Advice

- When learning a new command, check out the manpage to get a feel of what's available. Do not memorize the whole thing, this will happen with time automatically

Sections of a manpage

Section	Content
Name	Name & purpose of the command
Synopsis	Syntax of the command
Description	Full description of the command
Environment	Env variables related to the command
Author	Who done it
Files	Files related to the command
See Also	Other manual entries related to this command
Diagnostics	Documents status or error messages returned
Bugs	Known bugs

Searching through the online manual

- The online manual is divided into Sections
- The same entry may appear in multiple sections

The tools to search manpages generally allow you to specify the section you want to search so it's useful to know what they are

https://youtu.be/AS858I02Pzs

Sections of the online manual

Section #	Name	Notes
1	Commands	Not including bash built-ins (see help)
2	System calls	fork, execv, kill, …
3	Library Functions	printf, scanf…
4	Special Files	
5	File Formats	
6	Games	☺
7	Miscellaneous Information	
8	System Administration	mount, …

Intro pages & Specifying sections numbers

There are intro pages for each section

- `man intro` → manpage summarizing section 1 (commands)
- `man 1 intro`
- `man 2 intro` → system calls

So…We can specify section numbers!

- `man kill` → shows section 1 by default
- `man 2 kill` → shows manpage for system call

whatis

`whatis ls`

- → gives the short description that appears at top of manpage
- man ls → to show it

`whatis kill`

- → shows that kill appears in 2 sections of the manual

Does the same as `man –f`

- `man -f ls`
- `man -f kill`

apropos

`apropos`

- Searches for keywords in the one-line description of the command in the manpage

`apropos manual`

- Example: we get everything talking about the manual

Same as

- `man –k manual`

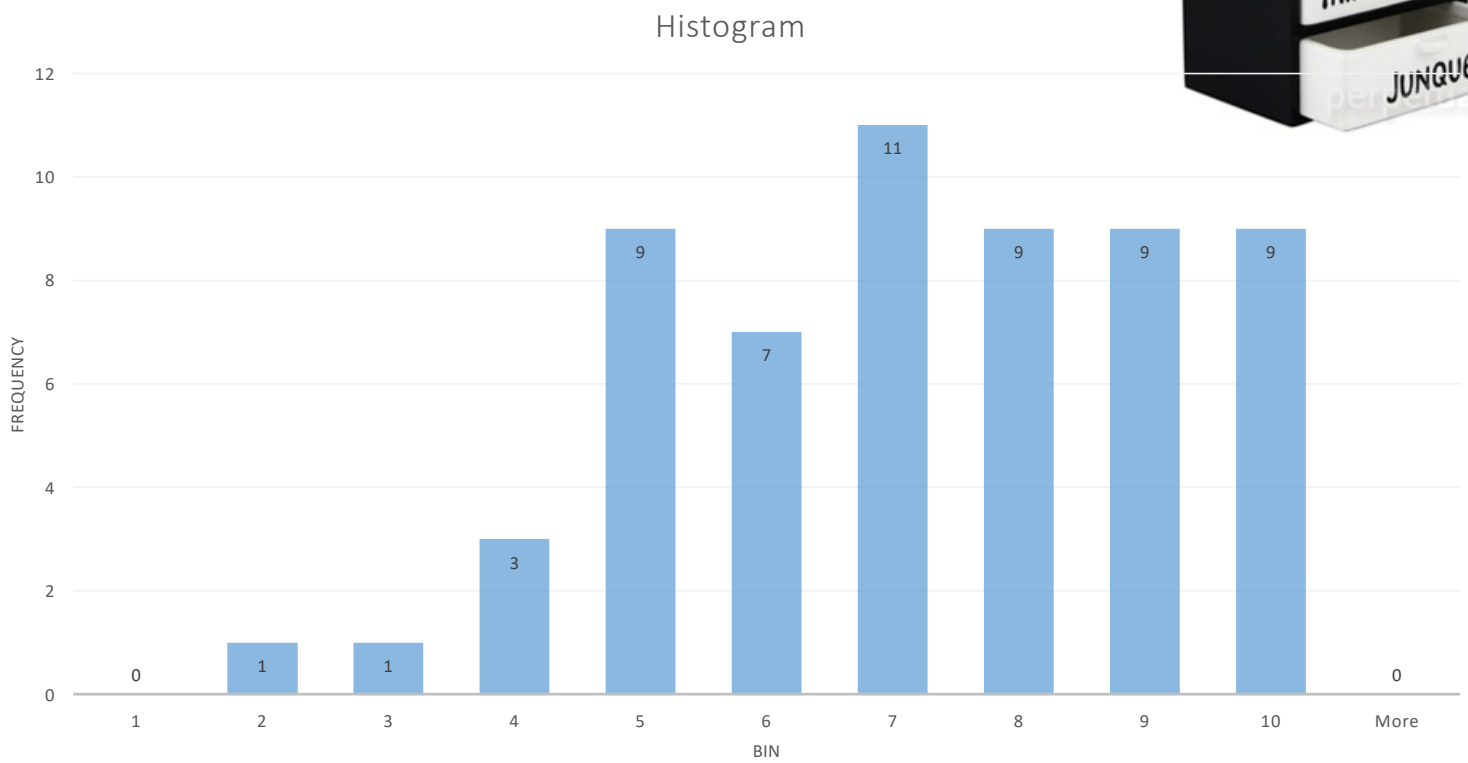
A quick apropos experiment :)

- `apropos list directory contents` → bunch of sols
- `apropos "list directly contents"` → only ls and related

- Same when searching with google
- This is due to
  - bash using spaces as separators between arguments to its commands already
  - so you need to enclose a string w/ spaces in "....", so that it's taken as 1 argument

Quick Announcements

- GQ-01 results are in!



Quick Announcements

IE1

- In-class on Wednesday 9/14/2022 @ usual class time
- Bring laptop + power plug
- Taken on canvas, proctored by Honorlock
- Bring your student ID!
- Covers everything so far, including what we will discuss about module M2 before the exam
- Slides allowed + Notes
- Ubuntu VM allowed

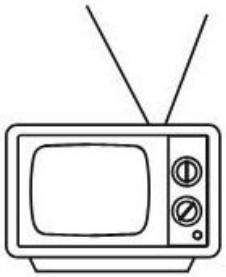
Quick Announcements

PA1

- To help you prepare / review for IE1
- Consider it a Q&A session with hands-on exercises
- Bring laptop + power plug
- Bring any questions you might have about the study material so far

PAs / IEs / Final Exam tentative schedule

Week #	Date	MON	WED
4	9/12	PA1	IE1
8	10/10	PA2	IE2
12	11/7	PA3	IE3
15	11/28	PA4	TBD
16	12/5	Final Exam	n/a



Previously On... CIS4930

- Basics of CLI filesystem navigation
  - pwd cd mkdir rmdir rm
  - dirs popd pushd
- Basics of CLI process management
  - ^C ^Z
  - fg bg jobs
  - Signals
- Getting help
  - Manpages structures & related tools

The basic toolkit to survive CLI

Module

M02
Serious CLI

Menu for this module

T1	Globbing	Bash allows you to use so-called meta-characters to build expressions allowing you to designate sets of filenames or folder names on which you may apply all sorts of CLI tools
T2	Shell Quoting & Escaping	One of the most interesting topics when learning Bash; the syntax allowing you to control the interpretation of the above-mentioned meta-characters or even substitute the result of executing code in an expression.
T3	Bash Environment, Variables, & Options	We then examine Bash options & variables.
T4	Bash Initialization Files	Finally, we are going to look at how we may configure the Bash shell for your user accounts. We will consider individual configuration files first, then system-wide ones.

Topic

M2T1
Globbing, Glob-Patterns, Filename Substitutions

Reading Assignment:

https://ryanstutorials.net/linuxtutorial/wildcards.php

List of globbing meta-characters

Meta character = character w/ special meaning to the shell

- Filename with . at start, or . after /, or just / → matched as is
- * → matches anything but dot as 1st character
- ? → matches any 1 character
- […] → single character alternatives
- [^…] → negation of the above
- {… , … , …} → multi-characters alternatives
- Begins with ~ → shorthand for homedir
- ! (…) → negate the enclosed globbing pattern

The Globbing Challenge

Use `touch` to create the following files:

file1	fileAB
file10	filea
file11	fileA
file2	fileAAA
File2	notAFile
File3	ThisOneEither5
file33	woohoo

https://linux-training.be/funhtml/ch17.html#idp54066976

1	List (with ls) all files starting with file
2	List all files containing File in their name
3	List (with ls) all files starting with file and ending in a number.
4	List (with ls) all files starting with file and ending with a lower case letter
5	List (with ls) all files starting with File and having a digit as fifth character.
6	List (with ls) all files starting with File and having a digit as fifth character and nothing else afterward.
7	List (with ls) all files starting with a lower case letter & ending w/ a digit.
8	List (with ls) all files that have exactly five characters.
9	List (with ls) all files that start with f or F and end with 3 or A.
10	List (with ls) all files that start with f have i or R as second character and end in a digit.
11	List all files that do not start with the letter F.
12	List all files that do not have File in their name

https://linux-training.be/funhtml/ch17.html#idp54066976

1	List (with ls) all files starting with file	ls file*
2	List all files containing File in their name	ls *File*
3	List (with ls) all files starting with file and ending in a number.	ls file*[0-9]
4	List (with ls) all files starting with file and ending with a lower case letter	ls file*[a-z]
5	List (with ls) all files starting with File and having a digit as fifth character.	ls File[0-9]*
6	List (with ls) all files starting with File and having a digit as fifth character and nothing else afterward.	ls File[0-9]
7	List (with ls) all files starting with a lower case letter & ending w/ a digit.	ls [a-z]*[0-9]
8	List (with ls) all files that have exactly five characters.	ls ?????
9	List (with ls) all files that start with f or F and end with 3 or A.	ls [fF]*[3A]
10	List (with ls) all files that start with f have i or R as second character and end in a digit.	ls f[iR]*[0-9]
11	List all files that do not start with the letter F.	ls [^F]*
12	List all files that do not have File in their name	ls !(*File*)

https://linux-training.be/funhtml/ch17.html#idp54066976

Wait! The last one is not working!

```
$ ls !(*File*)
bash: !: event not found
$ shopt extglob
extglob     off
$ shopt -s extglob
$ ls !(*File*)
ThisOneEither5    woohoo
$ shopt -u extglob
```

More about this when we cover Bash options

Want to read more about extended Globbing Patterns?

LINUX JOURNAL

https://www.linuxjournal.com/content/bash-extended-globbing

Examples of Extended Globbing

`?(pattern-list)`	Matches zero or one occurrence of the given patterns
`*(pattern-list)`	Matches zero or more occurrences of the given patterns
`+(pattern-list)`	Matches one or more occurrences of the given patterns
`@(pattern-list)`	Matches one of the given patterns
`!(pattern-list)`	Matches anything except one of the given patterns

Let’s try some of these!

- List all the JPEG and GIF files that start with either "ab" or "def”:

[redacted]

- How would we do that without extglob?

[redacted]

- List all the .jpg files that start with ab followed by one or more occurrences of the digit 2 or one or more occurrences of the digit 3

[redacted]

- How would we do that without extglob?

[redacted]

Let’s try some of these!

- List all the JPEG and GIF files that start with either "ab" or "def”:

```
ls +(ab|def)*+(.jpg|.gif)
```

- How would we do that without extglob?

```
ls ab*.jpg ab*.gif def*.jpg def*.gif
```

- List all the .jpg files that start with ab followed by one or more occurrences of the digit 2 or one or more occurrences of the digit 3

```
ls ab+(2|3).jpg
```

- How would we do that without extglob?

Nope :)

Actually, the above is more accurate. e.g., ababab.jpg @(ab|def) would be more in lines with the globbing

* Globbing is GREEDY

- list all the files that aren't JPEGs or GIFs

[redacted]

* Globbing is GREEDY

- list all the files that aren't JPEGs or GIFs

```
ls *!(.jpg|.gif)
```

- Doesn't work because the ".jpg" and the ".gif" of any file's name end up getting matched by the "*" and the *null string* at the end of the file name is the part that ends up *not* matching the "!(...)" pattern.

[redacted]

* Globbing is GREEDY

- list all the files that aren't JPEGs or GIFs

```
ls *!(.jpg|.gif)
```

- Doesn't work because the ".jpg" and the ".gif" of any file's name end up getting matched by the "*" and the *null string* at the end of the file name is the part that ends up *not* matching the "!(...)" pattern.

```
ls !(*.jpg|*.gif)
```

Topic

M2T2
Shell Quoting & Escaping

Reading Assignment:

https://ryanstutorials.net/linuxtutorial/wildcards.php

Bash Meta-Characters and Backslash Escaping

Trivial meta-character: SPACE → separates things in the CLI

- `touch filewith onespaceinitsname`
- `ls -l`

- `touch filewith\ onespaceinitsname`
- `ls –l`

It may mess w/ AUTOCOMPLETION

- `ls filewith [TAB]` → the space messes up the auto-completion
- `ls filewith\ [TAB]` → this works much better

…Works but tedious if we have many spaces…

(we'll see better later)

https://youtu.be/c457F9p7Gsw

Another silly example: \n meta-char

- echo hello world `[ENTER]`
- echo hello world \ `[ENTER]`

- Useless?
- Useful for multi-lines typing (convenience)

Escaping the \

- echo this is just a \\ in the command line

Escaping globbing meta-chars

- Setup
  - `touch COP2512 COP2513 COP4610 COP4931`
- Creating weird file or touch-ing the above folders?
  - `touch COP*`
  - `touch COP\*` → if I want a file with that weird name
- Same for removing
  - `rm COP*` → the COP* is erased, folders are safe but tried
  - `rm COP\*` → better; only file affect is COP*

Weird Case

What would happen

`touch COP*something`

- We expect that expansion / substitution would lead no results
  - Does that mean error?
  - Or we touch COPsomething?
  - Or we touch `[nothing at all]`

Weird Case

`touch COP*something`

- We expect that expansion / substation would lead no results

→ Because we have no results for the filename substitution
we keep the string COP*something **as is**

Why?

- By default, Bash expands a glob-pattern that matches nothing into itself

How to change this bash behavior?

- shopt -s nullglob

Other (related) bash options of interest

dotglob

- *If set, Bash includes filenames beginning with a '.' in the results of filename expansion.*
- *The filenames '.' and '..' must always be matched explicitly, even if* dotglob *is set.*

failglob

- *If set, patterns which fail to match filenames during filename expansion result in an expansion error.*

nocaseglob

- *If set, Bash matches filenames in a case-insensitive fashion when performing filename expansion.*

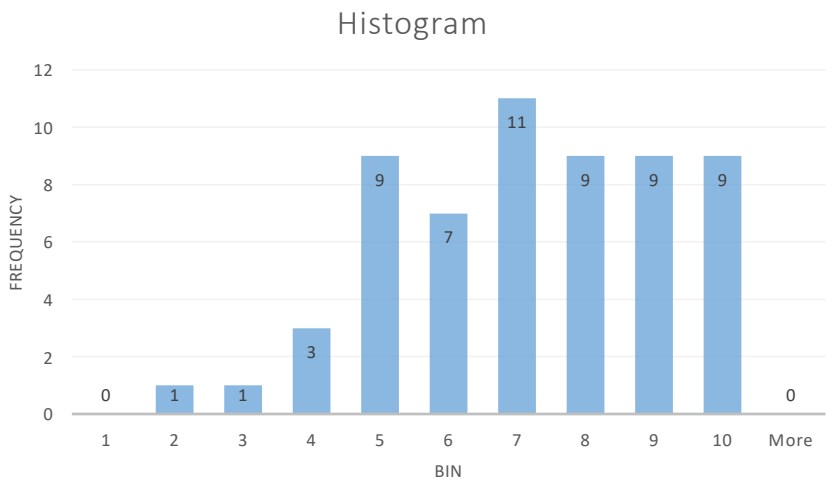
nullglob

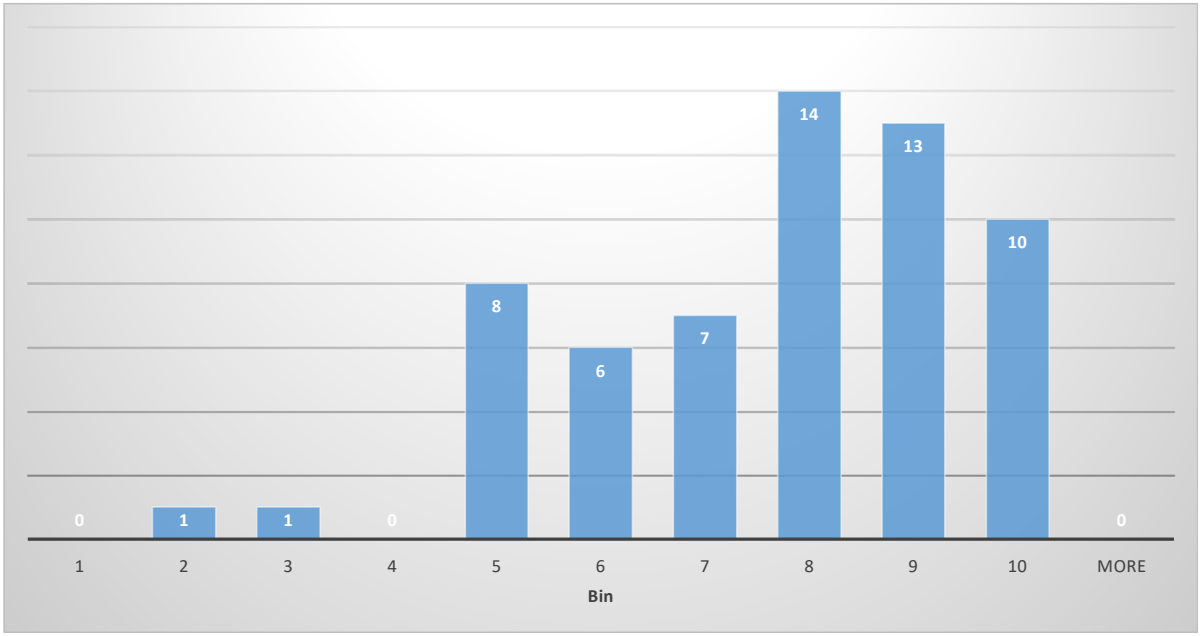
- *If set, Bash allows filename patterns which match no files to expand to a null string, rather than themselves.*

https://www.gnu.org/software/bash/manual/html_node/The-Shopt-Builtin.html

Quick Announcements

- GQ-01 updated results are in!
  - man –k / -f ambiguous “something”
  - alias alternative
- No late submissions





Actual footage showing Alessio explaining how `ls ab+(2|3)` works during last lecture.

+(.... | ....)

What it actually means:

- 1 or more occurrences of EITHER pattern
- +(aa|bb) would expand into aaaaaa bbbbbb but also aabbbbaa

What would we use instead in order to list only aaaaaa or bbbbbb but not aabbbbaa?

[redacted]

Topic

Interlude: PA1

These Practice Exercises are meant to help you review for our first Intermediary Exam: IE1.

Replacement Commands

You receive information that one of your servers was cracked, the cracker probably replaced the **ls** command. You know that the **echo** command is safe to use. Can **echo** replace **ls**? How can you list the files in the current directory with **echo**?

[redacted]

Is there another command, besides cd, to change directories?

[redacted]

Hacker Wannabe

It is a good day for a prank.

A friend left their workstation unlocked with a terminal running bash opened. We want to make it so that, when they use *ls*, the following message is displayed instead of running the command;

```
tux@tuxbox$ ls

ERROR 0xFF42 (bad sector \\ data corrupted).
Contact NSA for a "backup" copy of your data.
```

Aliases of aliases of….

- Define an alias `justdoit` that echo a message on the screen
- Define an alias `dontdoit` that calls the previous alias. What happens?
- Redefine the alias `justdoit` so that it calls the alias `dontdoit`. What happens?

Brace Yourselves: Defining multiple aliases to the same command

- We want to create aliases `one`, `two` and `three`, that all correspond to a `echo this is pretty cool`
- However, we want to issue only a single `alias` command

The Folders Factory

Create the following folders in one command each:

- Weird"folder
- Weird folder → yes, there is a space in the name of that one folder
- Weird\folder
- Weird/folder → yes, that's the actual name of the one folder
- Weird/folder → this time folder is a subfolder of Weird, which does not exist yet

If you bump into something impossible, explain why it is so.

All in one, and one in all!

We have a folder containing the following subfolders

- COP4610
- COP3353
- COP2512
- COP2513
- CIS4930

We want to move, in a single command, all folders except COP3353 into CIS4930. Do so without having to list explicitly all folders that you want to move.

Pause… Resume…

1. Execute a sleep 600 process in the background
2. Make sure it is running in the background
3. Find out its PID
4. Pause its execution by sending an appropriate signal to its PID
5. Display information confirming that it is now frozen
6. Resume its execution by sending an appropriate signal to its Job ID
7. Display information confirming that it is resumed
8. Ask it politely to terminate by sending it an appropriate signal

You may “quote me”

Provide the `echo` command you would type in your shell to display each of the following outputs;

- The dog & the cat; a tale of getting along just fine I remember now...
- "Bash is fun", they said. Liars!
- It's rather annoying (and even at times infuriating) to see quoting fail
- Why are we using \\ when we want just a \ to be displayed

Hidden treasures

- Let us list all files that end with the suffix .exe
- How do we make it so that the hidden files are also displayed?
- How do we make it so that only the hidden files matching the above pattern are displayed?

Ending with single digit

Let us list all files that start with a lowercase letter and end with a single digit:

Possible matches:

afilelikethis9

a9

Not matching:

Afilelikethis9

afilelikethis42

The case of the secret filename

How would you use globbing patterns to match filenames containing the word secret or SECRET?

Same question if the words can also be sEcReT and SeCrEt?

What if the spelling include all possible combinations of cases?

Extglob FTW!

Use extended globbing patterns to list the files that

- Start with the name of a color (red, blue, or green), followed by the name of an animal (cat, dog, wolf, panther)
- Contain 1 of more consecutive occurrences of a pattern consisting of the word project, followed by an underscore, followed by a 4 digit number.
- Are spelled using between 3 and 5 characters
- Start with the word number followed by a series of digits that are in ascending order. E.g., number123 would work, so would number233455. However, number231 would not be matched.

Topic

Interlude: PA1 - Solutions

These Practice Exercises are meant to help you review for our first Intermediary Exam: IE1.

+(.... | ....)

What it actually means:

- 1 or more occurrences of EITHER pattern
- +(aa|bb) would expand into aaaaaa bbbbbb but also aabbbbaa

What would we use instead in order to list only aaaaaa or bbbbbb but not aabbbbaa?

```
ls {+(aa),+(bb)}
ls @(+(aa)|+(bb))
```

Replacement Commands

You receive information that one of your servers was cracked, the cracker probably replaced the **ls** command. You know that the **echo** command is safe to use. Can **echo** replace **ls**? How can you list the files in the current directory with **echo**?

```
Echo *
```

Is there another command, besides cd, to change directories?

```
pushd or popd
```

Hacker Wannabe

It is a good day for a prank.

A friend left their workstation unlocked with a terminal running bash opened. We want to make it so that, when they use *ls*, the following message is displayed instead of running the command;

```
tux@tuxbox$ ls

ERROR 0xFF42 (bad sector \\ data corrupted).
Contact NSA for a "backup" copy of your data.
```

```
tux@tuxbox$ alias ls="echo ERROR 0xFF42 \(bad
sector \\\\ data corrupted\). Contact NSA for
a \"backup\" copy of your data."
```

Aliases of aliases of….

- Define an alias `justdoit` that echo a message on the screen
- Define an alias `dontdoit` that calls the previous alias. What happens?
- Redefine the alias `justdoit` so that it calls the alias `dontdoit`. What happens?

- The 1st two items work as expected; the 2nd alias calls the 1st that is replaced by its command
- The last item does not cause the infinite recursion that we might have expected

Brace Yourselves: Defining multiple aliases to the same command

- We want to create aliases `one`, `two` and `three`, that all correspond to a `echo this is pretty cool`
- However, we want to issue only a single `alias` command

```
• tux@tuxbox$ alias {one,two,three}='ls –l '
```

The Folders Factory

Create the following folders in one command each:

- Weird"folder
- Weird folder → yes, there is a space in the name of that one folder
- Weird\folder
- Weird/folder → yes, that's the actual name of the one folder
- Weird/folder → this time folder is a subfolder of Weird, which does not exist yet

If you bump into something impossible, explain why it is so.

```
tux@tuxbox$ mkdir Weird\"folder Weird\ folder
Weird\\folder
tux@tuxbox$ mkdir –p Weird/folder
```

All in one, and one in all!

We have a folder containing the following subfolders

- COP4610
- COP3353
- COP2512
- COP2513
- CIS4930

We want to move, in a single command, all folders except COP3353 into CIS4930. Do so without having to list explicitly all folders that you want to move.

```
tux@tuxbox$ mv !(CIS4930) CIS4930/
```

Pause… Resume…

1. Execute a sleep 600 process in the background
2. Make sure it is running in the background
3. Find out its PID
4. Pause its execution by sending an appropriate signal to its PID
5. Display information confirming that it is now frozen
6. Resume its execution by sending an appropriate signal to its Job ID
7. Display information confirming that it is resumed
8. Ask it politely to terminate by sending it an appropriate signal

```
sleep 600 &
jobs
ps or jobs –l
kill -s STOP 20599
jobs
kill -s CONT %1
jobs
kill -s TERM 20599
```

You may “quote me”

Provide the `echo` command you would type in your shell to display each of the following outputs;

- The dog & the cat; a tale of getting along just fine I remember now...
- "Bash is fun", they said. Liars!
- It's rather annoying (and even at times infuriating) to see quoting fail
- Why are we using \\ when we want just a \ to be displayed

```
echo ” \& \; \" \! \’ \( \) \\\\ \\"
```

Hidden treasures

- Let us list all files that end with the suffix .exe
- How do we make it so that the hidden files are also displayed?
- How do we make it so that only the hidden files matching the above pattern are displayed?

```
ls *.exe
shopt -s dotglob
ls *.exe
ls .*.exe
```

Ending with single digit

Let us list all files that start with a lowercase letter and end with a single digit:

Possible matches:

afilelikethis9

a9

Not matching:

Afilelikethis9

afilelikethis42

```
ls [a-z]*[^0-9][0-9]
```

The case of the secret filename

How would you use globbing patterns to match filenames containing the word secret or SECRET?

Same question if the words can also be sEcReT and SeCrEt?

What if the spelling include all possible combinations of cases?

```
shopt -s nocaseglob
ls –l *secret*
```

Extglob FTW!

Use extended globbing patterns to list the files that

- Start with the name of a color (red, blue, or green), followed by the name of an animal (cat, dog, wolf, panther)
- Contain 1 of more consecutive occurrences of a pattern consisting of the word project, followed by an underscore, followed by a 4 digit number.
- Are spelled using between 3 and 5 characters
- Start with the word number followed by a series of digits that are in ascending order. E.g., number123 would work, so would number233455. However, number231 would not be matched.

```
@(red|blue|green)@(cat,dog,wolf,panther)

+(project_[0-9][0-9][0-9][0-9])

@(???|????|?????)

Number*(1)*(2)*(3)*(4) and so on so forth
```

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.

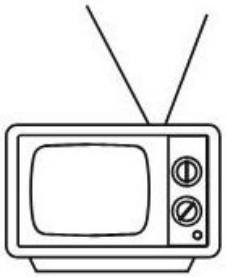
- Commands to be covered: diff, patch, sdiff

- 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