GlobalLogic

A Hitachi Group Company

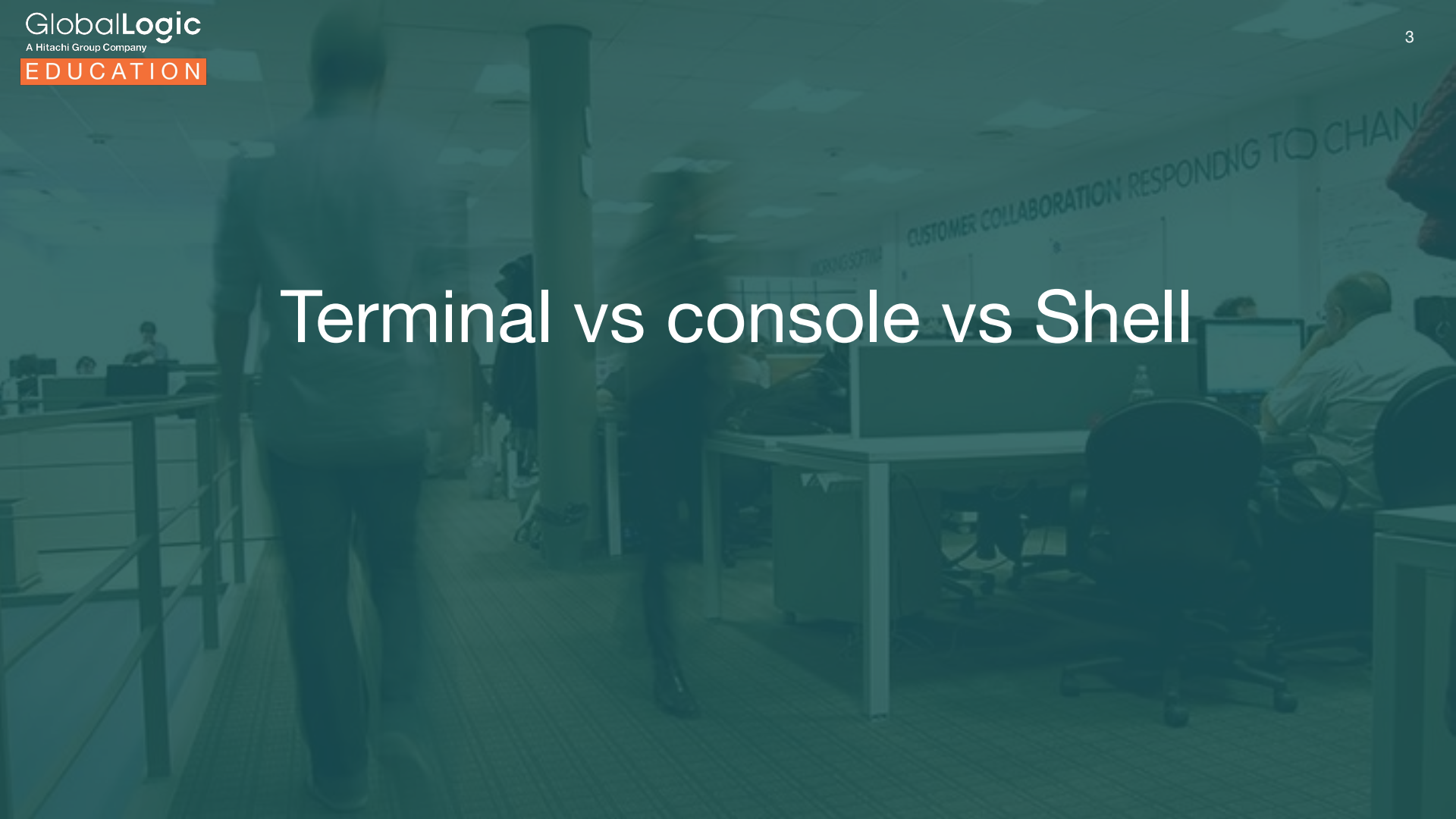
EDUCATION

Smart Start: Linux/Networking
Linux command line intro. Part 1

Sergii Kudriavtsev

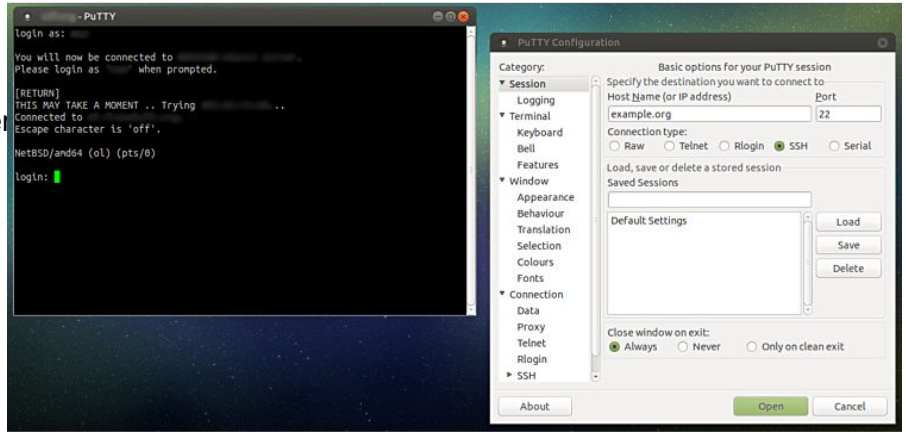
Agenda

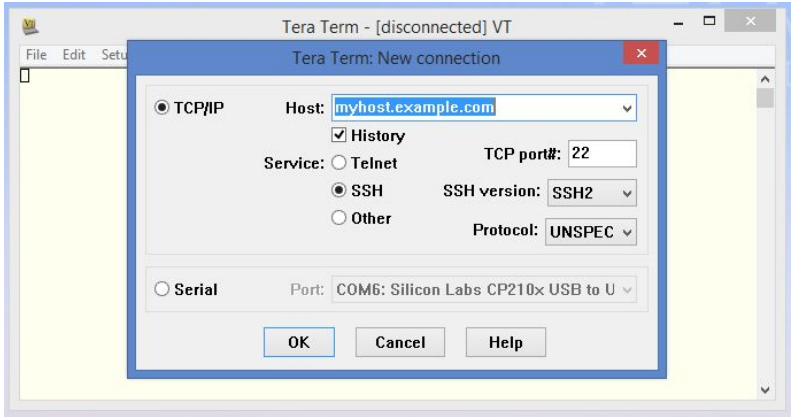
1. Terminal vs console vs Shell
2. Command structure. Starting in background
3. man
4. Most useful commands
5. package installation
6. Navigation
7. File operations
8. find
9. Escaping



Terminal vs console vs Shell

- Terminal vs console vs Shell
  - Terminal is a wrapper program which runs shell process. Terminal has connected device for input/output (putty, terra-ter minicom…)
    - `$ tty`
    - `/dev/tty*`
    - `/dev/pts/*`
  - Console is a physical terminal
    - Keyboard attached to the computer
    - Virtual consoles
  - Shell is a program which processes commands and returns output
- Local terminals (consoles)
  - ctrl+alt+F[1-6] Escape from GUI to Console
  - alt-F1, alt-F2, ..., alt-F6
  - alt-F7 - Get back to GUI Terminal





- Terminal settings
  - `$ reset # re-initialize terminal`
  - `$ stty # terminal settings`
  - `$ tty # print the terminal's device name`
    - `$ tty`
      `/dev/pts/14`
    - `$ echo 1234 > `tty` #or echo 1234> /dev/pts/14`
      `1234`
  - /dev/tty. This is a special file, representing the terminal for the current process
    - `$ echo 1234 > /dev/tty`
      `1234`
  - TERM env variable - Defines the termian type. Sets the terminal type for which output is to be prepared.
    - `$ export TERM=xterm # xterm, rxvt, konsole, vt100, linux e.t.c.`

The $TERM defines:

1. Length of lines
2. How to make the terminal beep
3. Is my terminal session is going to be colorful or not?
4. Specifies how your computer and the host computer to which you are connected exchange information.
5. All console applications such as vi/vim, ssh, and others dependent upon $TERM for communication and rendering information on a screen.

Shell

A shell is a command-line interpreter program that parses and sends commands to the operating system.

Types of Linux Shells
Linux offers different shell types for addressing various problems through unique features.

- Bash Shell - short for Bourne-Again SHell, is most commonly used shell in Linux.
- Zsh Shell. ...
- Fish Shell. ...
- Ksh Shell. ...
- Tcsh Shell. ...
- Dash Shell. ...
- Ash Shell. ...
- C Shell.

Bourne Shell (sh)

The Bourne shell was the first default shell on Unix systems, released in 1979. The shell program name is sh, and the traditional location is /bin/sh. The prompt switches to $, while the root prompt is #.

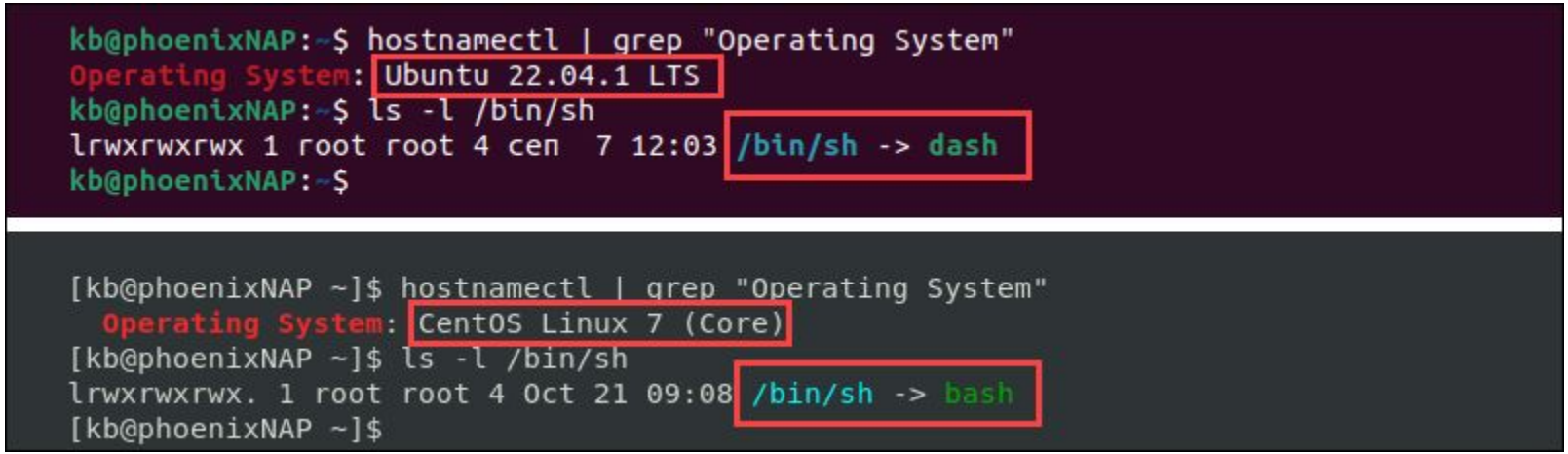
The Bourne shell quickly became popular because it is compact and fast. However, sh lacks some standard features, such as:

- Logical and arithmetic expansion.
- Command history.
- Other comprehensive features, such as autocomplete.

```
$ ps -p $$ -ocomm=
sh
$ foo=bar
$ echo $foo
bar
$ sudo -i
[sudo] password for kb:
# whoami
root
#
```

Modern Unix-like systems have the */bin/sh* executable file. The program does not start the Bourne shell but acts as an executable file pointing to the default system shell.

For most systems, the hard or symbolic link points to bash, while on Ubuntu and Debian, the link is to dash. In both cases, the link mimics the Bourne shell as much as possible.



C Shell (csh)

The C shell (csh) is a Linux shell from the late 1970s whose main objective was to improve interactive use and mimic the C language. Since the Linux kernel is predominantly written in C, the shell aims to provide stylistic consistency across the system.

The path to the C shell executable is */bin/csh*. The prompt uses `%` for regular users and `#` for the root user.

```
% ps -p $$ -ocomm=
csh
% set foo=bar
% echo $foo
bar
% sudo -i
[sudo] password for kb:
# whoami
root
#
```

New interactive features included:

- History of the previous command.
- User-defined aliases for programs.
- Relative home directory (~).
- Built-in expression grammar.

The main drawbacks of the C shell are:

- Syntax inconsistencies.
- No support for standard input/output (stdio) file handles or functions.
- Not fully recursive, which limits complex command handling.

The C shell improved readability and performance compared to the Bourne shell. The interactive features and innovations in csh influenced all subsequent Unix shells.

TENEX C Shell (tcsh)

The TENEX C shell (tcsh) is an extension of the C shell (csh) merged in the early 1980s. The shell is backward compatible with csh, with additional features and concepts borrowed from the TENEX OS.

The TENEX C shell executable path is in */bin/tcsh*. The user prompt is `hostname:directory>` while the root prompt is `hostname:directory#`. Early versions of Mac OS and the default root shell of FreeBSD use tcsh.

Additional features of the shell include:

- Advanced command history.
- Programmable autocomplete.
- Wildcard matching.
- Job control.
- Built-in where command.

Since tcsh is an extension of the C shell, many drawbacks persist in the extended version.

```
phoenixNAP:~> ps -p $$ -ocomm=
tcsh
phoenixNAP:~> set foo=bar
phoenixNAP:~> echo $foo
bar
phoenixNAP:~> sudo -i
[sudo] password for kb:
phoenixNAP:~# whoami
root
phoenixNAP:~#
```

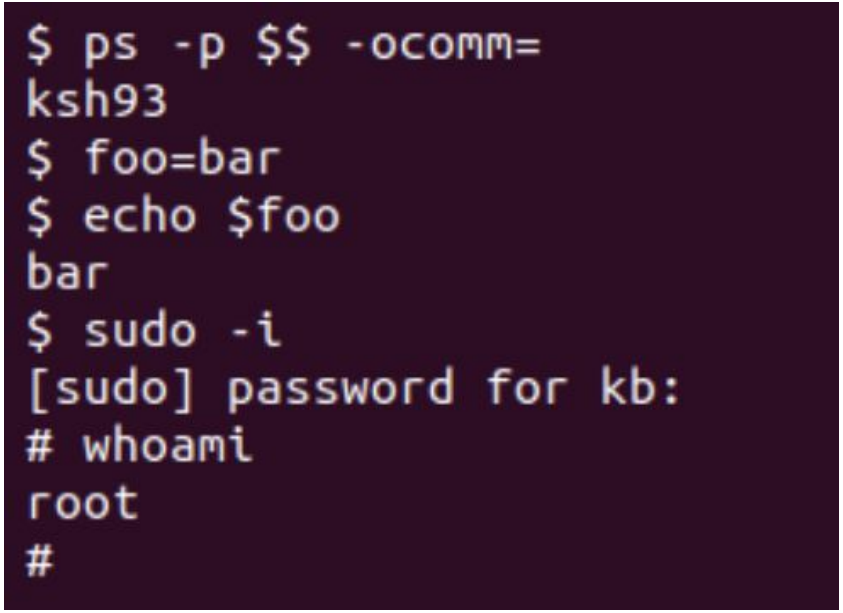
KornShell (ksh)

The KornShell (ksh) is a Unix shell and language based on the Bourne shell (sh) developed in the early 1980s. The location is in */bin/ksh* or */bin/ksh93*, while the prompt is the same as the Bourne shell (`$` for a user and `#` for root).

The shell implements features from the C shell and Bourne shell, aiming to focus on both interactive commands and programming features. The KornShell adds new features of its own, such as:

- Built-in mathematical functions and floating-point arithmetic.
- Object-oriented programming.
- Extensibility of built-in commands.
- Compatible with the Bourne shell.

The shell is faster than both the C shell and the Bourne shell.



Debian Almquist Shell (dash)

The Debian Almquist Shell (dash) is a Unix shell developed in the late 1990s from the Almquist shell (ash), which was ported to Debian and renamed.

Dash is famous for being the default shell for Ubuntu and Debian. The shell is minimal and POSIX compliant, making it convenient for OS startup scripts.

The executable path is */bin/dash*, in addition to */bin/sh* pointing to */bin/dash* on Ubuntu and Debian. The default and root user prompt is the same as in the Bourne shell.

Dash features include:

- Execution speeds up to 4x faster than bash and other shells.
- Requires minimal disk space, CPU, and RAM compared to alternatives.

The main drawback is that dash is not bash-compatible. The features not included in dash are known as "bashisms." Therefore, bash scripts require additional reworkings of bashisms to run succesfully.

```
$ ps -p $$ -ocomm=
dash
$ foo=bar
$ echo $foo
bar
$ sudo -i
[sudo] password for kb:
# whoami
root
#
```

Bourne Again Shell (bash)

The Bourne Again Shell is a Unix shell and command language created as an extension of the Bourne shell (sh) in 1989. The shell program is the default login shell for many Linux distributions and earlier versions of macOS.

The shell name shortens to bash, and the location is */bin/bash*. Like the Bourne shell, the bash prompt is `$` for a regular user and `#` for root.

Bash introduces features not found in the Bourne shell, some of which include:

- Brace expansion.
- Command completion.
- Basic debugging and signal handling.
- Command history.
- Conditional commands, such as the **bash if** and **bash case** statements.
- **Heredoc** support - a section of code that acts as a separate file. A HereDoc is a multiline string or a file literal for sending input streams to other commands and programs.

```
kb@phoenixNAP:~$ ps -p $$ -ocomm=
bash
kb@phoenixNAP:~$ foo=bar
kb@phoenixNAP:~$ echo $foo
bar
kb@phoenixNAP:~$ sudo -i
[sudo] password for kb:
root@phoenixNAP:~# whoami
root
root@phoenixNAP:~#
```

Z Shell (zsh)

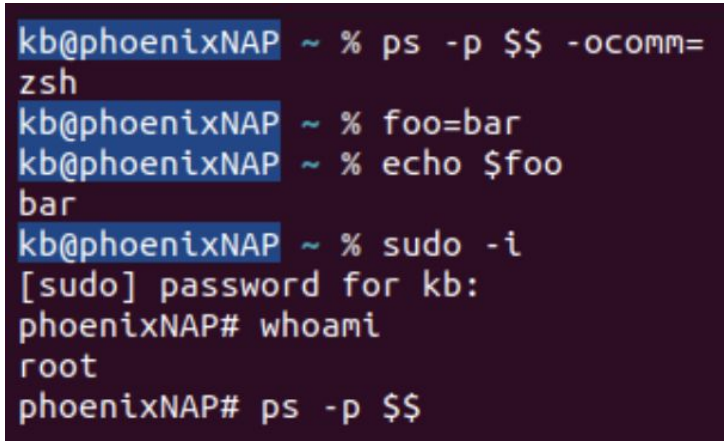
The Z shell (zsh) is a Unix shell created as an extension for the Bourne shell in the early 1990s. The feature-rich shell borrows ideas from ksh and tcsh to create a well-built and usable alternative.

The executable location is in */bin/zsh*. The prompt is `user@hostname location %` for regular users and `hostname#` for the root user. The Z shell is the default shell of Kali Linux and Mac OS.

Some new features added to the zsh include:

- Shared history among all running shell sessions.
- Improved array and variable handling.
- Spelling corrections and command name autofill.
- Various compatibility modes.
- Extensibility through plugins.

The shell is highly configurable and customizable due to the community-driven support through the **Oh My Zsh** framework.



Command structure.
Starting in background

Running Commands in Shell

- Command structure
  - `$ com`
  - `$ com arg1`
  - `$ com -a opt1 --long-flag opt2 -c opt3 arg1 arg2`
  - `$ com -a opt1 --long-flag opt2 -c opt3 arg1 arg2 > /dev/null`
  - `$ com -a opt1 --long-flag opt2 -c -- -opt3 arg1 arg2`
  - `$ com -a opt1 --long-flag opt2 -c - -opt3 arg1 arg2`
  - `$ [time] com`
  - `$ VARIABLE1=value1 VARIABLE2=value2 com arg1`
  - Example
    - `$ grep -r 1.txt`
    - `$ grep -- -r 1.txt #grep by data '-r'`
    - `$ grep '\-r' 1.txt`
    - `strings `which objdump` | grep \-`

Running in the background

- Detaching with &
  - `$ sleep 30 &`

Running Commands in Shell

- Commands processing
  - `$ type command_name`
  - Steps of Shell command processing:
    - Check if command name contains any slashes. If contains, locate utility on a file system.
      - /bin/date
      - ../date
    - Processing aliases and functions
      - `$ alias my_alias='echo "Inside alias"'`
      - `$ my_alias`
        `Inside alias`
      - `$ my_func(){ echo "Inside function"; }`
      - `$ my_func`
        `Inside function`

Running Commands in Shell

- Commands processing
  - Steps of Shell command processing:
    - Shell built-in
      - `$ man bash`
      - `$ type echo`
        `echo is a shell builtin`
      - `$ echo "message"`
        `message`
    - Hash tables
      - `$ hash`
      - `$ hash -r`
    - PATH variable - external utility on file system
      - `$ echo $PATH | tr ':' '\n'`
      - `$ which date`
        `/bin/date`
      - `$ date`
    - If everything above fails, command fails


man

- Manual
  - `$ man command`
    - `$ man kill`
  - `$ man page_num command`
    - `$ man 1 kill`
    - `$ man 2 kill`
  - `$ info command`
  - `$ man -k pattern`
  - `$ utility --help`

Most useful commands

- Commands to manage files and folders
  - `$ ls`
  - `$ pwd`
  - `$ cd`
  - `$ mkdir`
  - `$ cp`
  - `$ mv`
  - `$ rm`
  - `$ cat, zcat, less and more`
  - `$ ln`
  - `$ chmod`
  - `$ echo`
  - `$ find`
  - `$ grep`
  - `$ tar`
  - `$ touch`
  - `$ head`
  - `$ tail`

- Commands for managing users and groups
  - `$ adduser`
  - `$ groups`
  - `$ deluser`
  - `$ passwd`
  - `$ usermod`

- System Linux commands
  - `$ du`
  - `$ df`
  - `$ free`
  - `$ top`
  - `$ kill, pkill, and killall`
  - `$ mount`

Package installation

- dpkg, apt
  - Querying
    - `$ apt-cache policy # list of repositories`
    - `$ apt-cache pkgnames # list of packages installed`
    - `$ dpkg -l package_name # check if package installed`
    - `$ dpkg-query -L package_name # list files installed from the package`
  - Repository management (under root)
    - `/etc/apt/sources.list`
      - Comment out CD-ROM:
        - `# deb cdrom:[Ubuntu GNU/Linux 8.7.1 _Jessie_ - Official amd64 DVD Binary-1 20170116-11:01]/ jessie contrib main`
      - `$ apt-get update`
    - `$ sudo add-apt-repository 'deb http://httpredir.debian.org/debian jessie main'`
      - `$ apt-get update`
    - `$ sudo add-apt-repository ppa:yannubuntu/boot-repair`
      - `$ apt-get update`
  - Installation (under root)
    - `$ apt install package_name`
    - `$ apt-get upgrade`
    - `$ apt-get upgrade package_name`

Navigation

- Navigation
  - `$ ls # print list of files and directories under current working directory`
    - `/`
    - `$ ls file1 file2 dir1 # list file1, file2 and contents of dir1`
    - `$ ls -l # use a long listing format`
    - `$ ls -A, $ ls -a # show also files with names starting from point: .profile, .kshrc`
    - `$ ls -t # sort by time of file modification: most recent files first`
    - `$ ls -r # sort output in reverse order`
    - `$ ls -R # recursive`
    - `$ ls -i # print inodes`
    - `$ ls -d # print the name of directory`
    - `$ ls -S # sort by file size`
  - `$ pwd # print current working directory`
  - `$ cd directory # change current working directory to directory`
    - `$ cd - # return to previous directory`
    - `$ cd .. # go 1 level up`
    - `$ cd, cd ~ # go to the home directory`

File operations

- Information
  - `$ file file.txt`
- Text files
  - viewing
    - `PAGER` env variable
      - `$ export PAGER=less`
    - `$ cat file.txt # print file on stdout`
    - `$ less file.txt # view the file in pager`
      - Flags
        - -N
      - Shortcuts
        - help: h
        - movement: g, G, 125g, p, 50p
        - search: /, ?
          - /value
          - /(word1|word2)
          - /(word1|word2) word3
            - word1 word3|word2 word3
        - exit: q

- Information
  - `$ file file.txt`
- Text files
  - viewing
    - ~~`$ more file.txt # view the file in pager`~~
    - `$ head file.txt # print first 10 lines of file on stdout`
    - `$ head -n 20 file.txt # print first 20 lines of file on stdout`
    - `$ tail file.txt # print last 10 lines of file on stdout`
    - `$ tail -n 20 file.txt # print last 20 lines of file on stdout`
    - `$ tail -f file.txt # output on stdout appended data as the file1.txt grows`
    - `$ tail -f file1.txt | tee -a file2.txt # output on stdout appended data as the file1.txt grows and saving the output in file2.txt`
    - `$ diff file1.txt file2.txt`
    - `$ diff -c3 file1.txt file2.txt`
      - `$ diff -c3 file1.txt file2.txt > 1.diff`
      - `$ mcedit 1.diff`
    - `$ comm -1 -2 file1.txt file2.txt`

- editing
  - `EDITOR` env variable
    - `$ export EDITOR=mcedit`
  - `$ vi file.txt`
    - editing
      - I, i, A, a
      - x, dw, db, dd
      - Esc
    - searching
      - /pattern
      - ?pattern
    - saving
      - :w
    - exiting
      - :q
      - :q!
      - :x
      - :wq
    - key bindings
      - http://www.viemu.com/vi-vim-cheat-sheet.gif
  - `$ mcedit file.txt # needs custom installation. Package "mc"`
  - `$ nano file.txt # may be not installed`
  - ~~`$ emacs file.txt # do not try this at home`~~

- Binaries
  - `$ strings binary`
  - `$ mcedit binary`
  - `$ hexdump binary`
  - `$ hexdump -n 304 -C binary`
  - `$ hexdump -n 304 -C blk00000.datXXX coreutil`

```
00000000  f9 be b4 d9 1d 01 00 00  01 00 00 00 00 00 00 00  |................|
00000010  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 3b a3 ed fd  |............;...|
00000030  7a 7b 12 b2 7a c7 2c 3e  67 76 8f 61 7f c8 1b c3  |z{..z.,>gv.a....|
00000040  88 8a 51 32 3a 9f b8 aa  4b 1e 5e 4a 29 ab 5f 49  |..Q2:...K.^J).._I|
00000050  ff ff 00 1d 1d ac 2b 7c  01 01 00 00 00 01 00 00  |......+|........|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 ff ff  |................|
00000080  ff ff 4d 04 ff ff 00 1d  01 04 45 54 68 65 20 54  |..M.......EThe T|
00000090  69 6d 65 73 20 30 33 2f  4a 61 6e 2f 32 30 30 39  |imes 03/Jan/2009|
000000a0  20 43 68 61 6e 63 65 6c  6c 6f 72 20 6f 6e 20 62  | Chancellor on b|
000000b0  72 69 6e 6b 20 6f 66 20  73 65 63 6f 6e 64 20 62  |rink of second b|
000000c0  61 69 6c 6f 75 74 20 66  6f 72 20 62 61 6e 6b 73  |ailout for banks|
000000d0  ff ff ff ff 01 00 f2 05  2a 01 00 00 00 43 41 04  |........*....CA.|
000000e0  67 8a fd b0 fe 55 48 27  19 67 f1 a6 71 30 b7 10  |g....UH'.g..q0..|
000000f0  5c d6 a8 28 e0 39 09 a6  79 62 e0 ea 1f 61 de b6  |\..(.9..yb...a..|
00000100  49 f6 bc 3f 4c ef 38 c4  f3 55 04 e5 1e c1 12 de  |I..?L.8..U......|
00000110  5c 38 4d f7 ba 0b 8d 57  8a 4c 70 2b 6b f1 1d 5f  |\8M....W.Lp+k.._|
00000120  ac 00 00 00 00 f9 be b4  d9 d7 00 00 00 01 00 00  |................|
00000130
```


find

- $ find
  - $ find *directory*
  - $ find
  - $ find -name *file.txt*
  - $ find *directory* -name *file.txt*
  - $ find *directory* -name *'*.txt'*
  - $ find *directory* -name *'file.???'*
  - $ find *directory* -name *'file.**'*
  - $ find *directory* -name *'file.[a-z]'*
  - $ find *directory* -path '*bin'
  - $ find *directory* -type f
  - $ find *directory* -exec grep "*regexp*" {} \;
  - $ find *directory* -exec ls -l {} \;
  - $ find *directory* -exec ls -l {} \+
  - $ find *directory* -type d -or -name '*b*'

- constructing with xargs
  - $ find ~ -name '*.txt' | xargs tar cvf archive_text.tar
    - $ tar cvf archive_text.tar `find ~ -name '*.txt'`
  - $ find ~ -name '*.tmp' | xargs rm -f
    - $ rm -f `find ~ -name '*.tmp'`

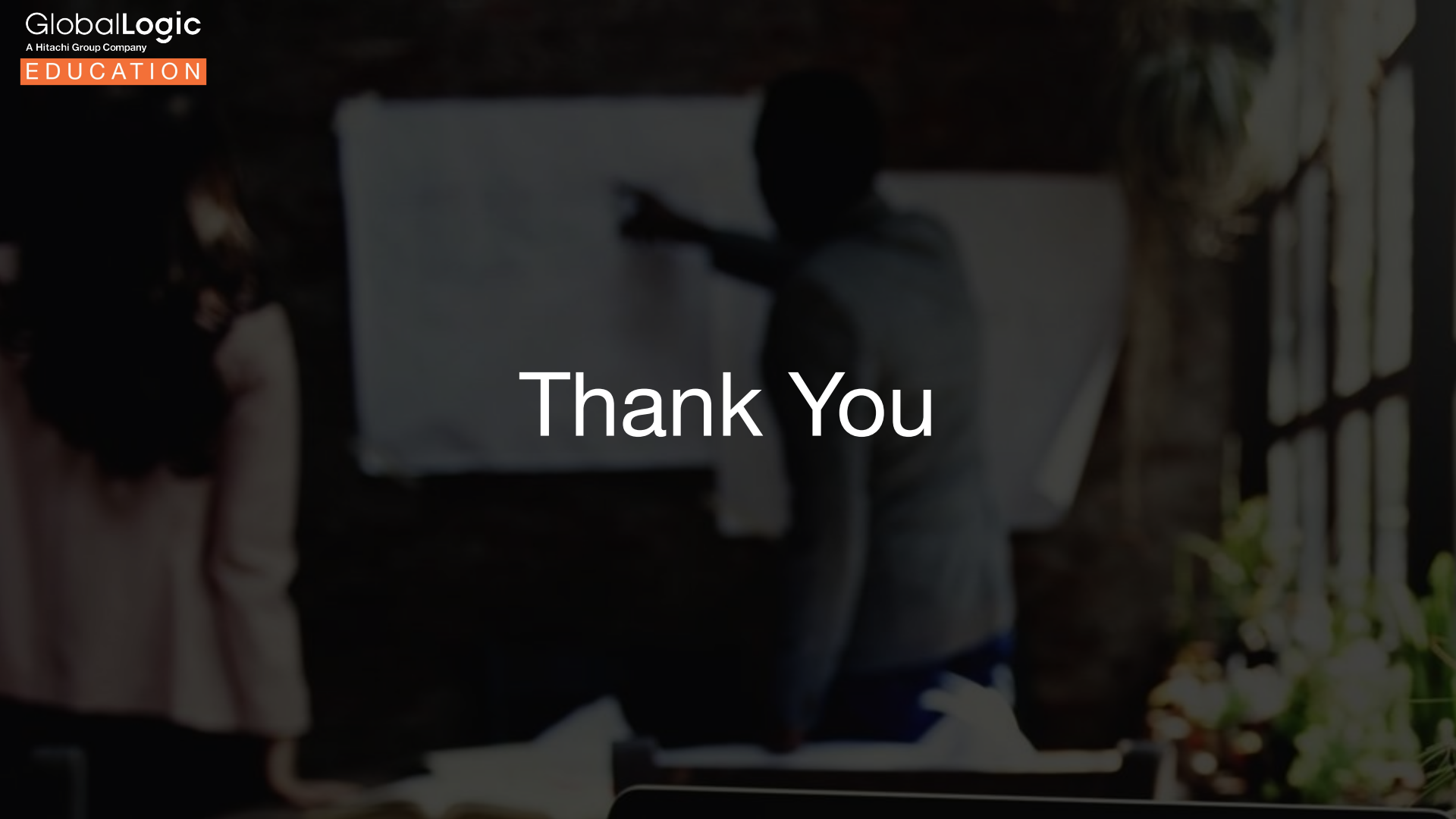
Escaping

- Escaping in Shell
  - Special characters
    - `$ man bash`
    - space, tabulation
    - $, *, [,] and so on
  - Escaping
    - \ (back slash)
      - `$ ls -l my\ directory`
      - `$ DIR=my\ directory; ls -l "$DIR"`
    - ' (single quote)
      - `$ ls -l 'my directory'`
      - `$ DIR='my directory'; ls -l '$DIR' # ls -l  '$DIR' will not work`
      - special characters are not processed
    - " (double quote)
      - `$ ls -l "my directory"`
      - `$ DIR="my directory"; ls -l "$DIR"`
      - special characters to be processed:
        - $
        - ` (backtick)

- Escaping in Shell
  - Examples

```
echo -e "\n\v\b\a"
```

\ n	newline (newline)
\ r	a carriage return
\ t	tabulation
\ v	vertical tab
\ b	backspace
\ a	"bell" (signal)
\ 0xx	


GlobalLogic
A Hitachi Group Company
EDUCATION
Thank You