Advanced Console Techniques

Command Sequences. Streams. Text Editing. Searching for and Within Files. SUDO Management

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.bg

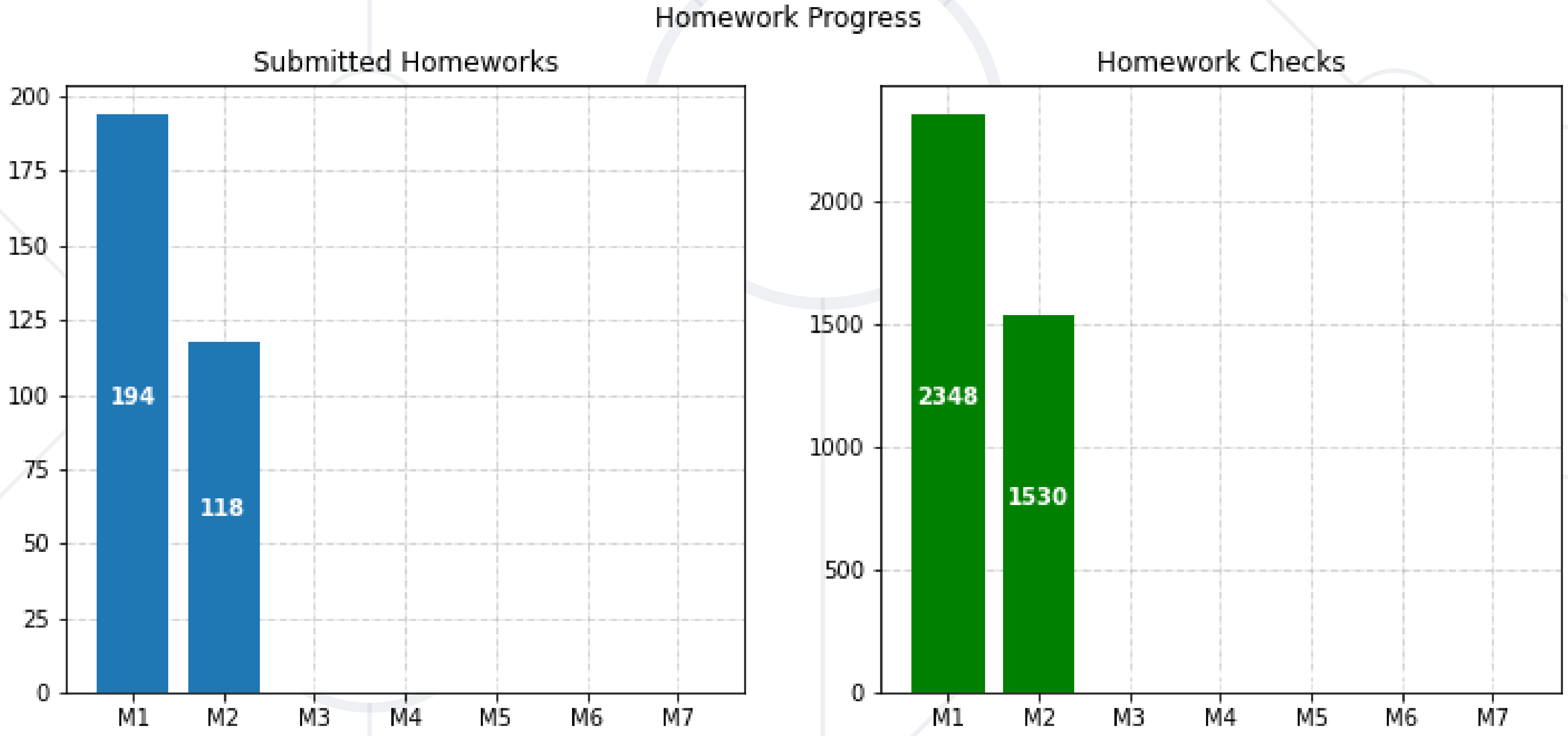
Have a Question?

Software University

sli.do

#LSA

Homework Progress



Solutions for **M2** can be submitted until **23:59:59** on **20.03.2025**

Solutions for **M3** can be submitted until **23:59:59** on **27.03.2025**

Previous Module (M2)

Quick Overview

What We Covered

1. Console Deep Dive
2. Getting Help
3. Files and Folders
4. Users and Groups
5. Access Rights

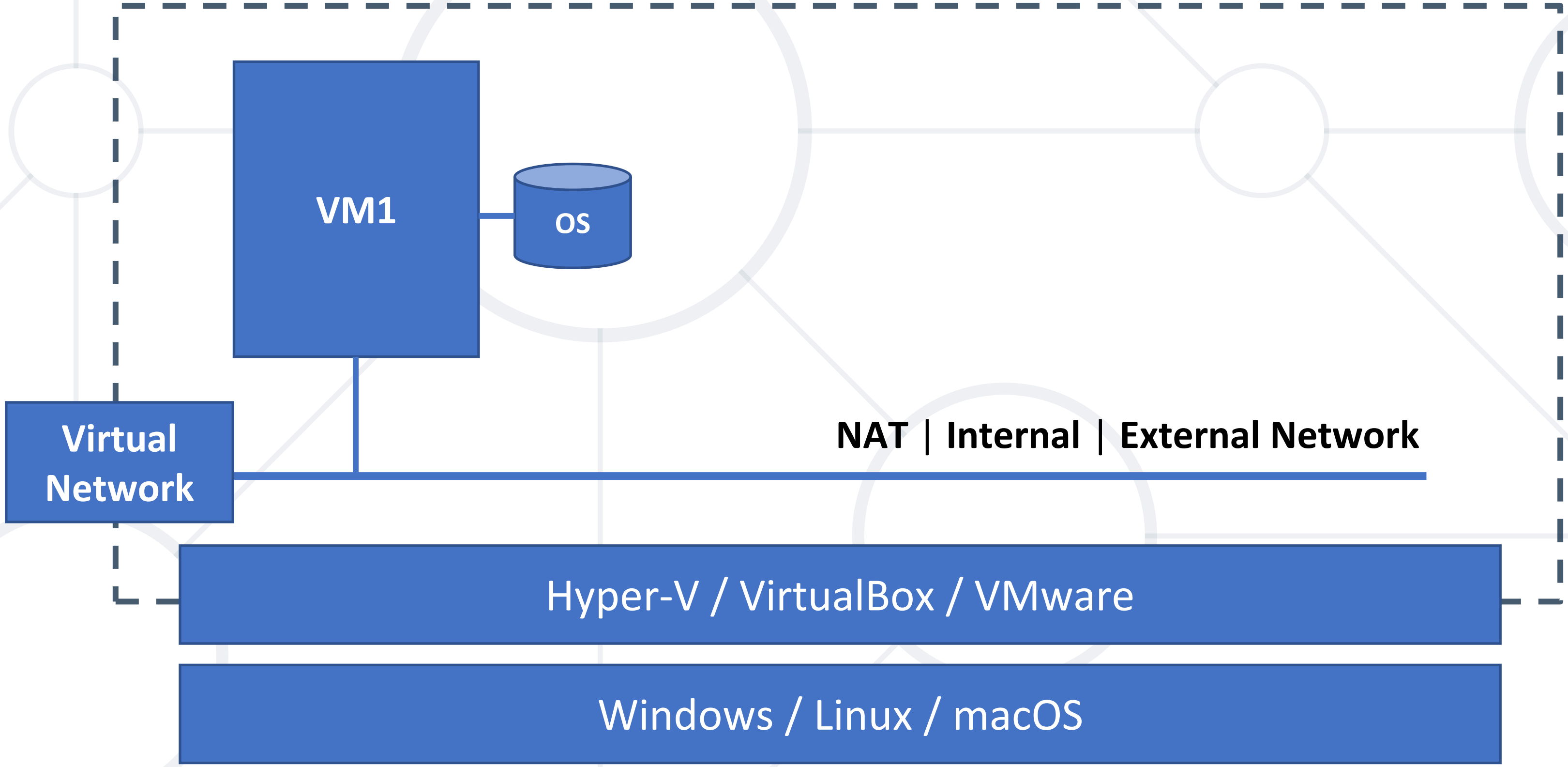
This Module (M3)

Topics and Lab Infrastructure

Table of Contents

1. Input / Output Streams
2. Command Sequences
3. Regular Expressions
4. Advanced File Techniques
5. Screen Editors
6. SUDO Management

Lab Infrastructure

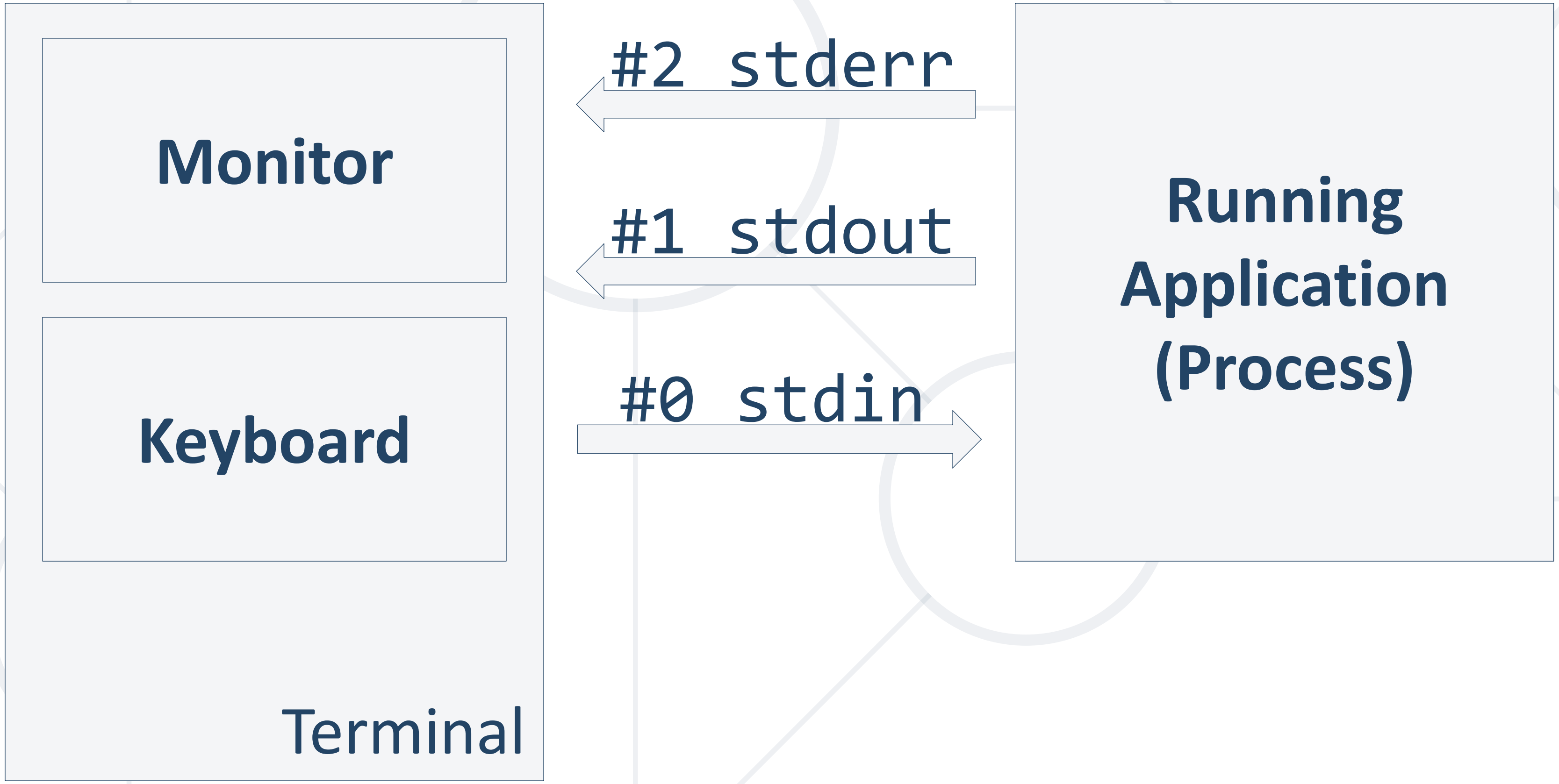

VM1
OS
Virtual Network
NAT | Internal | External Network
Hyper-V / VirtualBox / VMware
Windows / Linux / macOS

```
stdin
stdout
stderr
```

Input / Output Streams

Standard File Descriptors. Redirection

Standard File Descriptors



Redirect Input (<)

- Description
  - Redirect input stream (**stdin**). Usually, it is omitted
- Example

```
[user@host ~]$ cat < hello.txt
Hello!
...
[user@host ~]$ cat hello.txt
Hello!
...
```

Redirect Output (>)

- Description
  - Redirect output streams (**stdout** or **stderr**) with target overwrite
- Example

```
[user@host ~]$ echo 'Hello!' > hello.txt
...
[user@host ~]$ echo 'Hello!' 1> hello.txt
...
[user@host ~]$ cat hello.txt
Hello!
...
```

Redirect Output with Append (>>)

- Description
  - Redirect output streams (**stdout** or **stderr**) with target append
- Example

```
[user@host ~]$ cat file.txt
Line #1
[user@host ~]$ echo 'Line #2' >> file.txt
[user@host ~]$ cat file.txt
Line #1
Line #2
```

Set -/+o Noclobber

- Description
  - (Dis)allow existing regular files to be overwritten by redirection
- Example

```
[user@host ~]$ set -o noclobber
[user@host ~]$ echo 'Hi!' > file.txt
[user@host ~]$ echo 'Hi!' > file.txt
bash: file.txt: existing file cannot be overwritten
[user@host ~]$
```

Redirection Order

- Order is important
  - Redirection instructions are **processed left to right**
- Example

```
[user@host ~]$ cat missing.txt > out.txt 2>&1

# is different compared to this

[user@host ~]$ cat missing.txt 2>&1 > out.txt
```

Redirection Recipes

- Only stdout

```
[user@host ~]$ ls -alF > dir_list.txt
```

- Both stdout and stderr – different targets

```
[user@host ~]$ ls -al file.txt > ok.txt 2> err.txt
```

- Both stdout and stderr – same target

```
[user@host ~]$ ls -al file.txt > res.txt 2>&1
```

Problem: Create Document

- Description
  - Create text document on the command line (on the fly)
- Example

```
[user@host ~]$ cat file.txt
Line #1
Line #2
Line #3
```

Solution(s): Create Document

- Solution #1 (heredoc)

```
[user@host ~]$ cat > file.txt << EOF
>Line #1
>Line #2
>Line #3
>EOF
[user@host ~]$ cat file.txt
Line #1
Line #2
Line #3
```

- Solution #2 (echo)

```
[user@host ~]$ echo 'Line #1' > file.txt
[user@host ~]$ echo 'Line #2' >> file.txt
[user@host ~]$ echo 'Line #3' >> file.txt
[user@host ~]$ cat file.txt
Line #1
Line #2
Line #3
```

- Same result different approaches

Command Sequences

Execute Multiple Commands. Substitution

Commands Sequences

- Execute in order (disconnected)
  - **Sequence**: **command1 ; command2**
- Execute in order (connected)
  - **Pipe**: **command1 | command2**
- Execute conditionally
  - On **Success**: **command1 && command2**
  - On **Failure**: **command1 || command2**

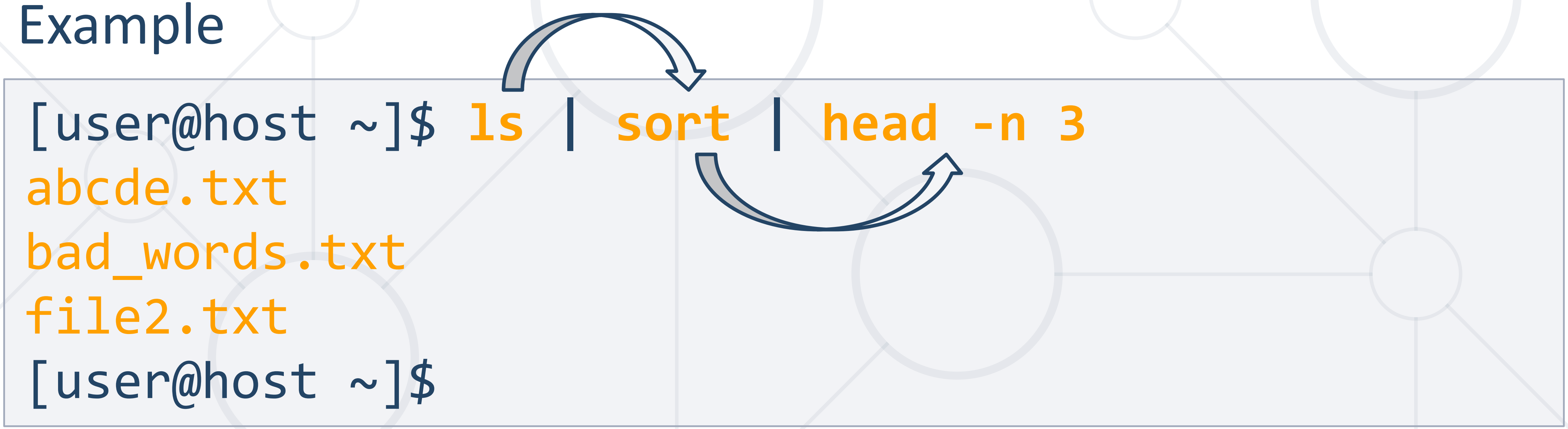
Sequence (;)

- Description
  - Always execute next command
- Example

```
[user@host ~]$ ls non-existing-file.txt ; echo Ok
ls: cannot access non-existing-file.txt: No such
file or directory
Ok
[user@host ~]$
```

Pipe (|)

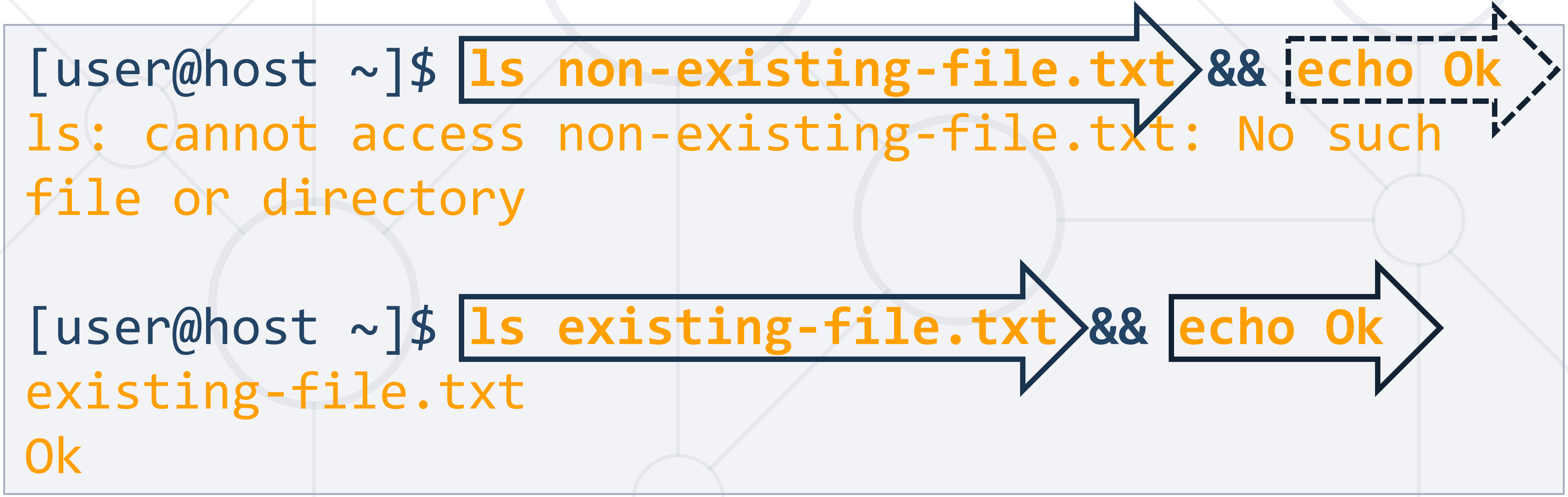
- Description
  - Chaining two or more programs' output together
- Example



```
[user@host ~]$ ls | sort | head -n 3
abcde.txt
bad_words.txt
file2.txt
[user@host ~]$
```

On Success (&&)

- Description
  - Next command is executed if previous one exited with status of 0
- Example

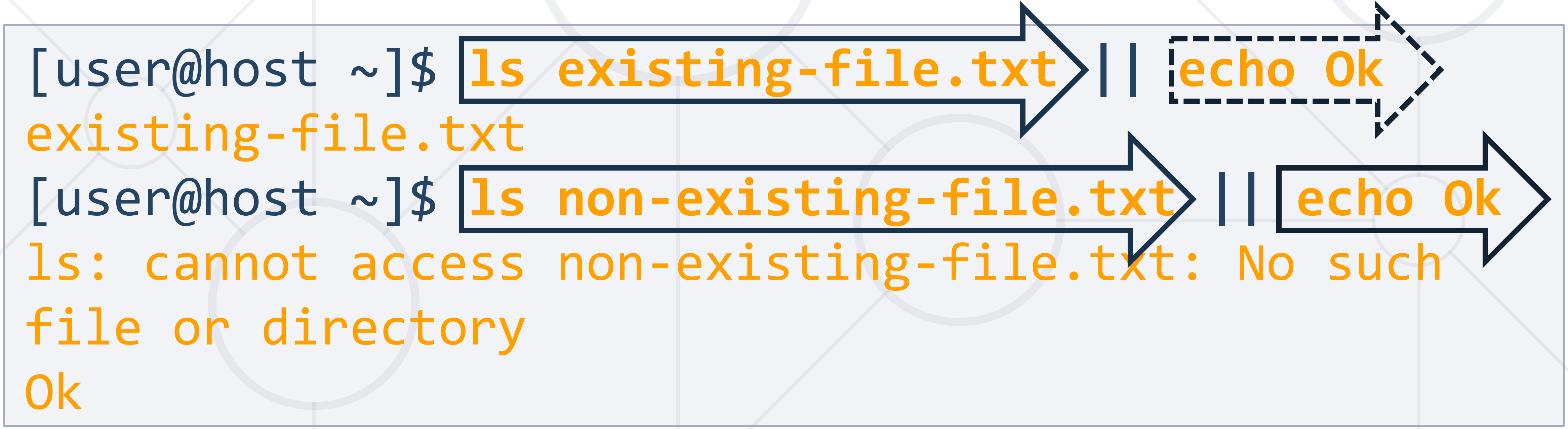


```
[user@host ~]$ ls non-existing-file.txt && echo Ok
ls: cannot access non-existing-file.txt: No such
file or directory

[user@host ~]$ ls existing-file.txt && echo Ok
existing-file.txt
Ok
```

On Failure (||)

- Description
  - Next command is **NOT** attempted if previous one exited with 0
- Example



```
[user@host ~]$ ls existing-file.txt || echo Ok
existing-file.txt
[user@host ~]$ ls non-existing-file.txt || echo Ok
ls: cannot access non-existing-file.txt: No such
file or directory
Ok
```

Command Substitution

- Description
  - Substitute the command output for the command itself
- Example

```
# file_name.txt contains the text /etc/os-release
[user@host ~]$ cat `cat file_name.txt`
...

[user@host ~]$ cat $(cat file_name.txt)
...
```

Breaking Long Commands

- Instead of having this

```
[user@host ~]$ cut -d : -f 7 /etc/passwd | sort |
uniq | wc -l
```

- We could do it this way*

```
[user@host ~]$ cut -d : -f 7 /etc/passwd \
               | sort \
               | uniq \
               | wc -l
```

* A prompt managed by the **$PS2** environment variable will appear on the multiline commands, asking us to continue entering the command

tee

- Purpose
  - Read from standard input and write to standard output and files
- Syntax

```
tee [options] [file]
```

- Examples

```
# Show file content on screen and save it to file
[user@host ~]$ cat list.txt | tee listed.txt
# List directory on the screen and append to file
[user@host ~]$ ls -al / | tee -a root-dir.txt
```

xargs

- Purpose
  - Build and execute command lines from standard input
- Syntax

```
xargs [options] [command [initial arguments]]
```

- Examples

```
# Delete list of files read from a text file
[user@host ~]$ cat file_list.txt | xargs rm -rf
# Show file content of every *.conf file in /etc
[user@host ~]$ ls /etc/*.conf | xargs cat
```

Practice

`^[abc]\.??`

Regular Expressions

Know them. Use them

Wildcards

- *
  - Any characters
- ?
  - Any single character
- [characters]
  - Any character that is a member of the set *characters*
- [!characters]
  - Any character that is not a member of the set *characters*
- [[:class:]]
  - Any character that is a member of the specified *class*

Character Classes

- Frequently used

Class	Description
[:alnum:]	Alphanumeric characters A-Z, a-z, and 0-9
[:word:]	Same as [:alnum:] including underscore (_)
[:alpha:]	Alphabetic characters A-Z and a-z
[:digit:]	Numeric characters 0-9
[:lower:]	All lowercase letters a-z
[:upper:]	All uppercase letters A-Z

Globing Examples

File names are **case sensitive**!

- *****
  - All files
- **a***
  - Any file beginning with ***a***
- **a*.txt**
  - Any file beginning with ***a*** and ending with ***.txt***
- **[abc]???**
  - Any file beginning with either ***a***, ***b***, or ***c***, and followed by ***3 chars***

Bracket Expressions

- Inclusion (names starting with a, b, or c)

```
[user@host ~]$ ls [abc]*.txt
```

- Exclusion (names that DO NOT start with a, b, or c)

```
[user@host ~]$ ls [^abc]*.txt
```

- Ranges (names starting with any symbol between a and z)

```
[user@host ~]$ ls [a-z]*.txt
```

Regular Expressions

- Consists of **literals** and **metacharacters**
- Basic Regular Expressions (BRE)
  - **^** , **$** , **.** , **[** , **]** , *****
- Extended Regular Expressions (ERE)
  - BRE + **(** , **)** , **{** , **}** , **?** , **+** , **|**

Control Characters

- .
  - any single character - (.text) => atext, btext2, 2text, …
- ^
  - Begging of the line - (^text) => text, textone, texttwo, …
- $
  - End of the line - (text$) => text, newtext, lasttext, …
- \
  - Escape character - (.\.text) => new.text, new.text2, …

Quantifiers

- **?**
  - Match an element ***zero*** times or ***one*** time
- *****
  - Match an element ***zero*** or ***more*** times
- **+**
  - Match an element ***one*** or ***more*** times
- **{}**
  - Match an element a ***specific number*** of times

Rule	Meaning
{n}	Exactly n times
{n,m}	At least n times, but not more than m times
{n,}	n or more times
{,m}	No more than m times

grep

- Purpose
  - Print lines matching a pattern
- Syntax

```
grep [options] patterns [files]
```

- Examples

```
# Display lines containing the false word #1
[user@host ~]$ grep -n false /etc/passwd
# Display lines containing the false word #2
[user@host ~]$ cat /etc/passwd | grep -n false
```

A Few Usage Scenarios

- Display all lines starting with ***one*** or ***two***

```
[user@host ~]$ grep -E '^(one|two)' list.txt
```

- Display all lines starting with ***one*** or containing ***two***

```
[user@host ~]$ grep -E '^one|two' list.txt
```

- Display all lines containing first ***one*** and then ***two***

```
[user@host ~]$ grep -E 'one.*two' list.txt
```

- Display all lines containing ***one*** and ***two*** in any order

```
[user@host ~]$ grep one list.txt | grep two
```

Advanced File Techniques

Find them. Work with them

find

- Purpose
  - Search for files in a directory hierarchy
- Syntax

```
find [options] [starting point] [expression]
```

- Examples

```
# Find all *.txt files starting from current dir
[user@host ~]$ find . -type f -name *.txt
# Search for files executable by others
[user@host ~]$ find . -type f -perm /o+x
```

Common Find Scenarios

- All files owned by particular user

```
[root@host ~]# find /tmp -type f -user root
```

- All files that do not belong to particular user

```
[root@host ~]# find /tmp -type f ! -user root
```

- All files bigger than 10 MB

```
[root@host ~]# find / -type f -size +10M -ls
```

- All files changed today

```
[root@host ~]# find /tmp -type f -mtime 0 -ls
```

locate*

- Purpose
  - Find files by name
- Syntax

```
locate [options] pattern
```

- Examples

```
# Locate all readme files
[root@host ~]# locate readme
# Locate all readme files in a case insensitive way
[root@host ~]# locate -i readme
```

* It is not installed automatically in every distribution. You may have to install it additionally

updatedb*

- Purpose
  - Update a database for mlocate
- Syntax

```
updatedb [options]
```

- Examples

```
# Update the database
[root@host ~]# updatedb
# Write the update to a file
[root@host ~]# updatedb -o output.txt
```

* It is not installed automatically in every distribution. You may have to install it additionally

Extract Data

Extract Data from Files. Combine Files

more

- Purpose
  - A filter for paging through text one screen at a time
- Syntax

```
more [options] [files]
```

- Examples

```
# Open one file for reading
[user@host ~]$ more /etc/services
# Open two files for reading
[user@host ~]$ more /etc/os-release /etc/services
```

less*

- Purpose
  - It is similar to **more**, but allows movement in both directions
- Syntax

```
less [options] [files]
```

- Examples

```
# Open one file for reading
[user@host ~]$ less /etc/services
# Open two files for reading
[user@host ~]$ less /etc/os-release /etc/services
```

* A common joke says that **less is more** ☺

head

- Purpose
  - Output the **first part** (***10 lines by default***) of files
- Syntax

```
head [options] [files]
```

- Examples

```
# Show first ten lines of a file
[user@host ~]$ head /etc/passwd
# Show first three lines of a file
[user@host ~]$ head -n 3 /etc/passwd
```

tail

- Purpose
  - Output the **last part** (***10 lines by default***) of files
- Syntax

```
tail [options] [files]
```

- Examples

```
# Show last ten lines of a file
[user@host ~]$ tail /etc/passwd
# Show last three lines of a file
[user@host ~]$ tail -n 3 /etc/passwd
```

tac

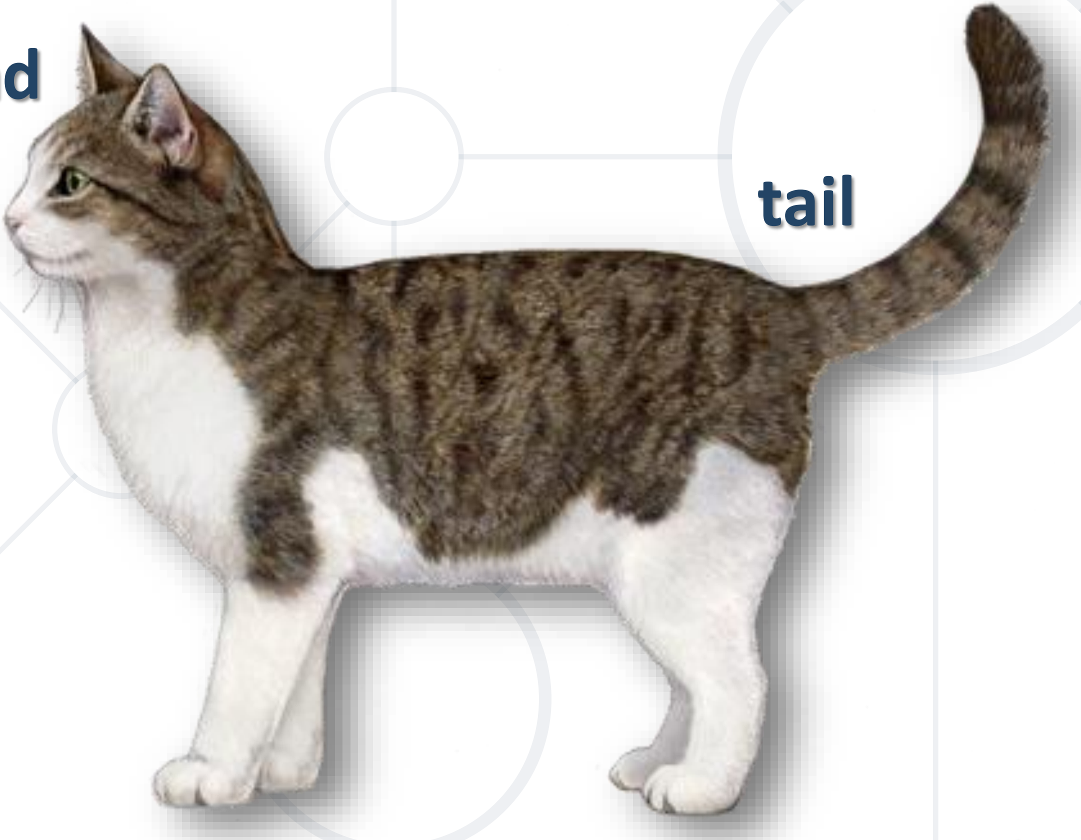
- Purpose
  - Concatenate and print files in reverse
- Syntax

```
tac [options] [files]
```

- Examples

```
# Print one file in reverse
[user@host ~]$ tac readme.txt
# Print /etc/*.conf files in reverse
[user@host ~]$ tac /etc/*.conf
```

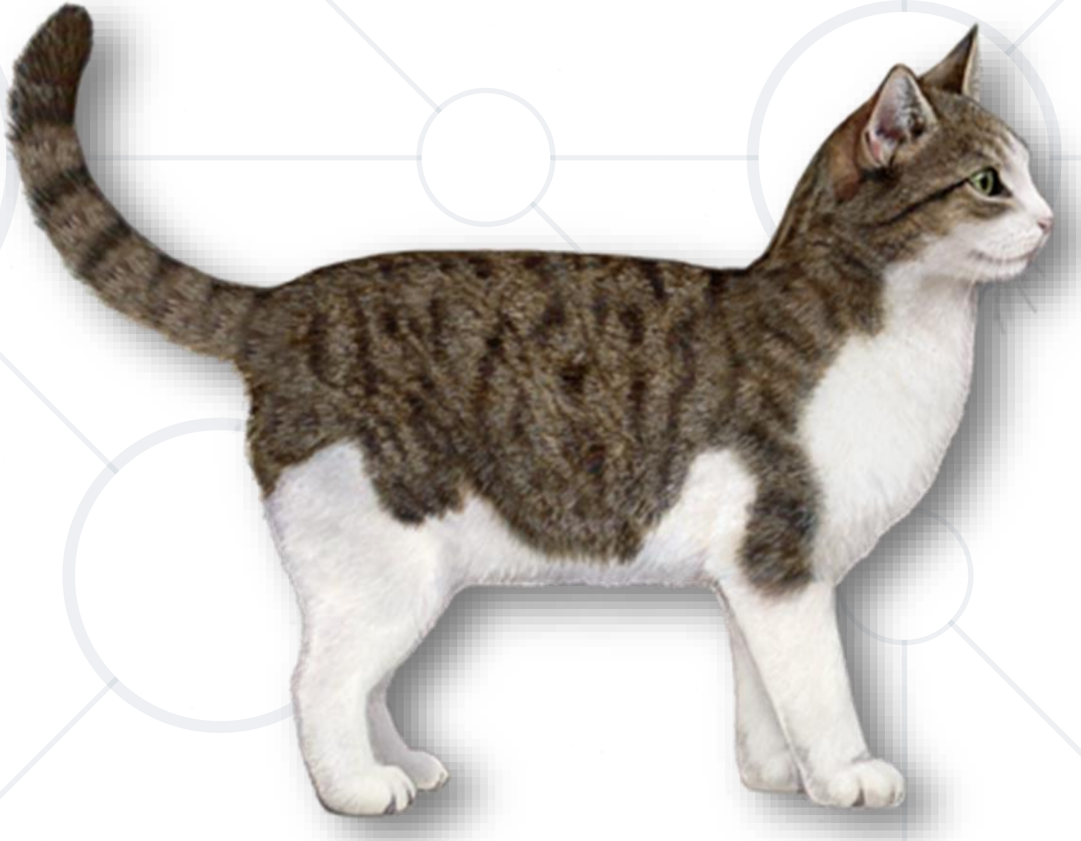
Visual Explanation ☺



head

tail

cat



tac

https://www.wisdompanel.com/en-us/cat-breeds/domestic-cat-finland

uniq

- Purpose
  - Report or omit repeated lines
- Syntax

```
uniq [options] [files]
```

- Examples

```
# Print only duplicate lines
[user@host ~]$ uniq -D file.txt
# Print contents with repeated lines omitted
[user@host ~]$ uniq file.txt
```

sort

- Purpose
  - Sort lines of text files
- Syntax

```
sort [options] [files]
```

- Examples

```
# Print sorted content of a file
[user@host ~]$ sort file.txt
# Print sorted only unique lines of a file
[user@host ~]$ sort -u file.txt
```

wc

- Purpose
  - Print newline, word, and byte counts for each file
- Syntax

```
wc [options] [files]
```

- Examples

```
# Print statistics for a file
[user@host ~]$ wc /etc/service
# Print number of newlines in a file
[user@host ~]$ wc -l /etc/service
```

nl

- Purpose
  - Add number to the beginning of every line in a file
- Syntax

```
nl [options] [files]
```

- Examples

```
# Print numbered lines read from a file
[user@host ~]$ nl /etc/service
# Print numbered lines with leading zeroes
[user@host ~]$ nl -w 4 -nrz /etc/service
```

cut

- Purpose
  - Remove sections from each line of files
- Syntax

```
cut options [files]
```

- Examples

```
# Cut field #1 (username) from /etc/passwd
[user@host ~]$ cut -d : -f 1 /etc/passwd
# Cut fields #1 and #7 from /etc/passwd
[user@host ~]$ cut -d : -f 1,7 /etc/passwd
```

paste

- Purpose
  - Merge lines of files
- Syntax

```
paste [options] [files]
```

- Examples

```
# Merge two files
[user@host ~]$ paste day_num.txt day_name.txt
```

join

- Purpose
  - Join lines of two files on a common field
- Syntax

```
join [options] file1 file2
```

- Examples

```
# Join two files
[user@host ~]$ join -t : -j 1 f1.txt f2.txt
```

split

- Purpose
  - Split a file into pieces
- Syntax

```
split [options] [input [prefix]]
```

- Examples

```
# Split file in multiple files 50 lines each
[user@host ~]$ split -l 50 services
# Split file in multiple files 50 lines each #2
[user@host ~]$ split -a 3 –d -l 50 services part
```

expand

- Purpose
  - Convert tabs to spaces
- Syntax

```
expand [options] [files]
```

- Examples

```
# Convert tabs to four spaces each
[user@host ~]$ expand -t 4 file.txt
```

unexpand

- Purpose
  - Convert spaces to tabs
- Syntax

```
unexpand [options] [files]
```

- Examples

```
# Convert every four spaces to tab
[user@host ~]$ unexpand -t 4 file.txt
```

fmt

- Purpose
  - Provides simple text formatting
- Syntax

```
fmt [options] [files]
```

- Examples

```
# Format the text to 60 columns
[user@host ~]$ fmt --width 60 file.txt
```

tr

- Purpose
  - Translate or delete characters
- Syntax

```
tr [options] set1 [set2]
```

- Examples

```
# Convert every : to |
[user@host ~]$ tr ':' '|' < /etc/passwd
# Delete all occurrences of :
[user@host ~]$ tr -d ':' < /etc/passwd
```

od

- Purpose
  - Dump files in octal or other formats
- Syntax

```
od [options] [files]
```

- Examples

```
# Print file’s content in octal format
[user@host ~]$ od /etc/passwd
# Print file’s content using named characres
[user@host ~]$ od -a /etc/passwd
```

Practice

Screen Editors

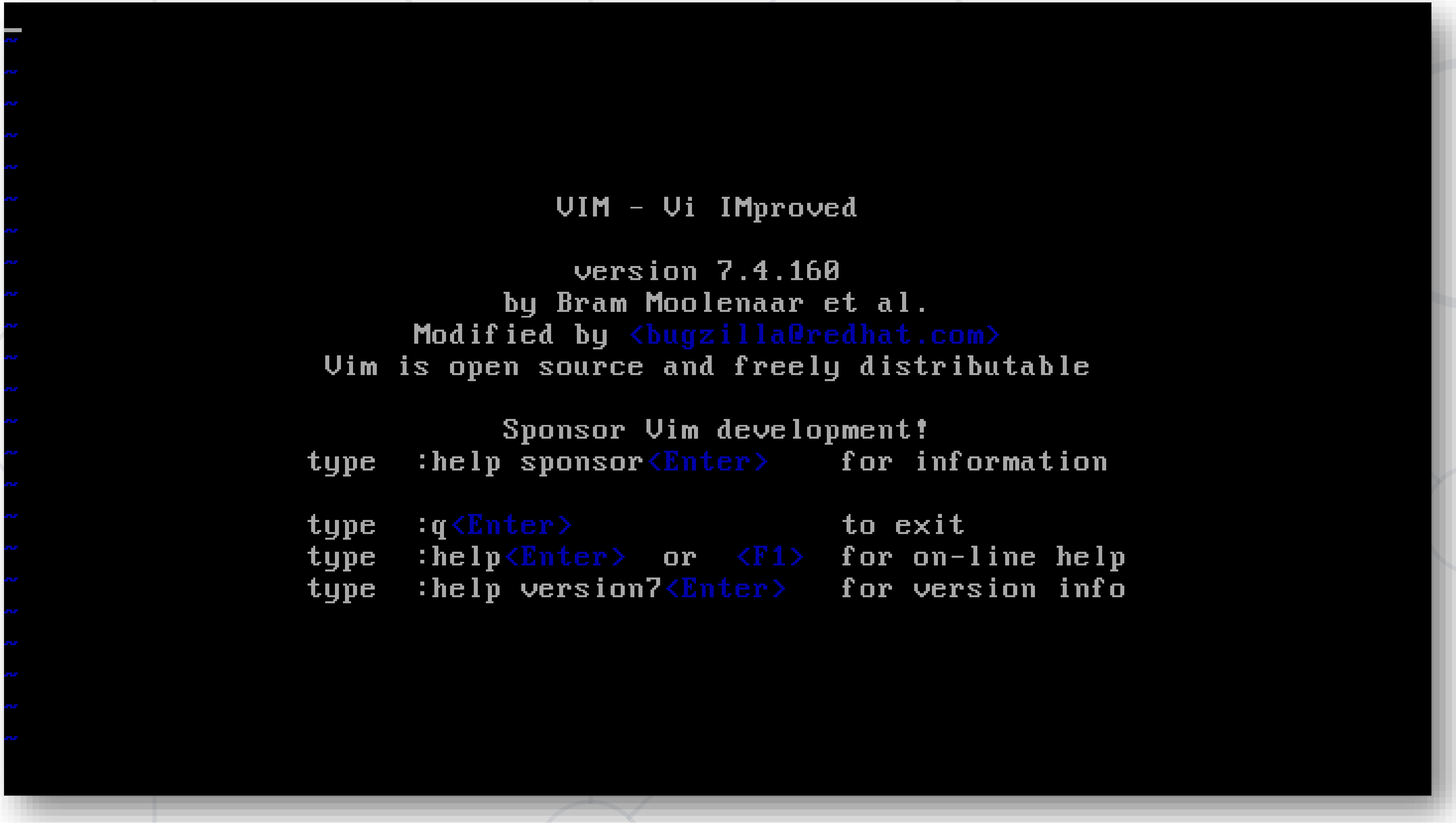
Characteristics. vim

Screen Editors*

- Characteristics
  - File content is seen one screen at a time
  - Offer content navigation (line-by-line and page-by-page)
  - Commands are invoked from a menu or key combinations
  - Offer syntax highlighting, line numbering, etc.
  - User experience varies
- Typical Screen Editors
  - vi (vim), nano, pico, joe, emacs

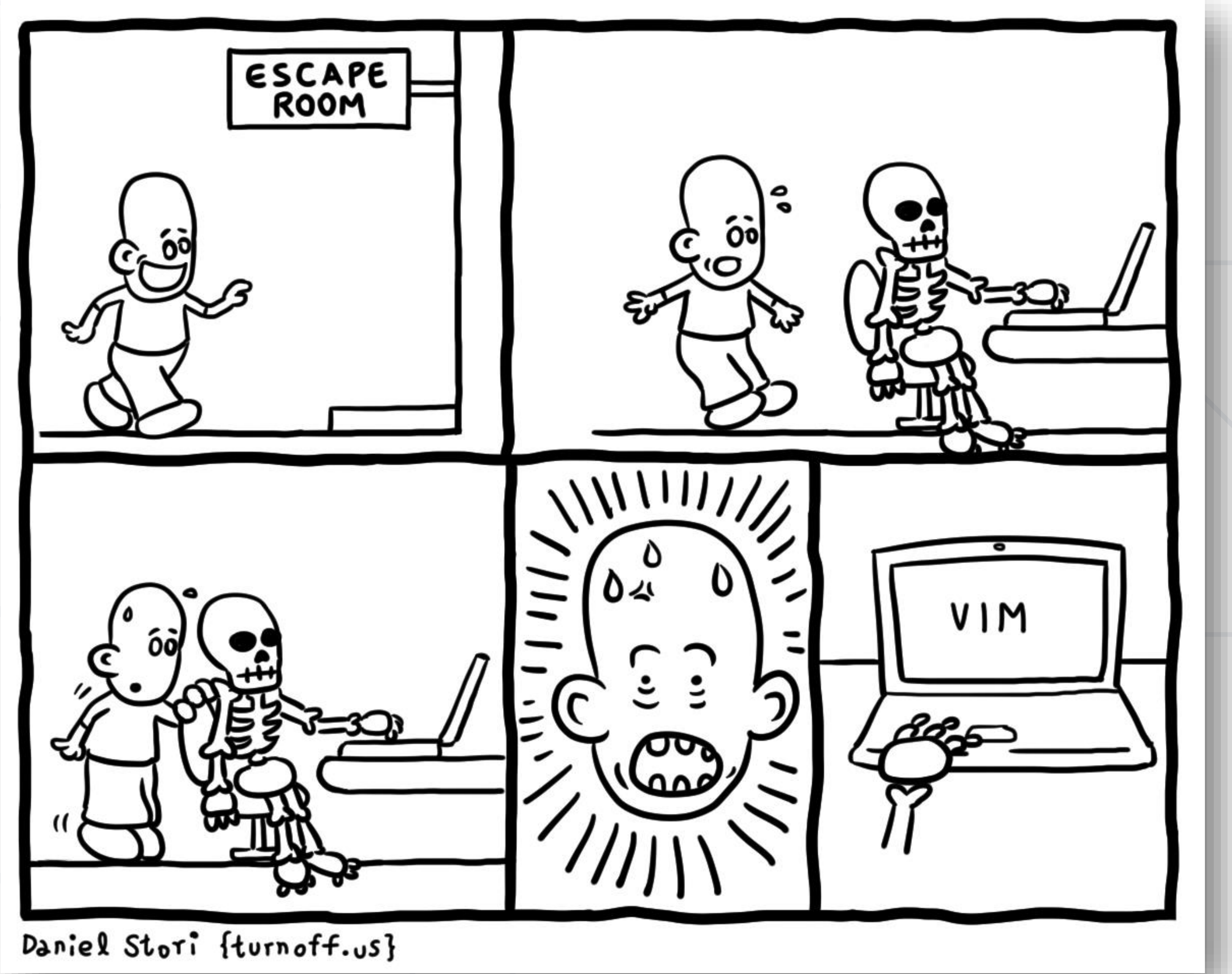
* Screen text editor available by default varies between distributions

VIM (Vi IMproved)



```
~
~
~
~
~
~                         VIM - Vi IMproved
~
~                          version 7.4.160
~                     by Bram Moolenaar et al.
~              Modified by <bugzilla@redhat.com>
~           Vim is open source and freely distributable
~
~                     Sponsor Vim development!
~          type  :help sponsor<Enter>    for information
~
~          type  :q<Enter>               to exit
~          type  :help<Enter>  or  <F1>  for on-line help
~          type  :help version7<Enter>   for version info
~
~
~
~
~
```

Quit VIM If You Can



http://turnoff.us/geek/escape-room/

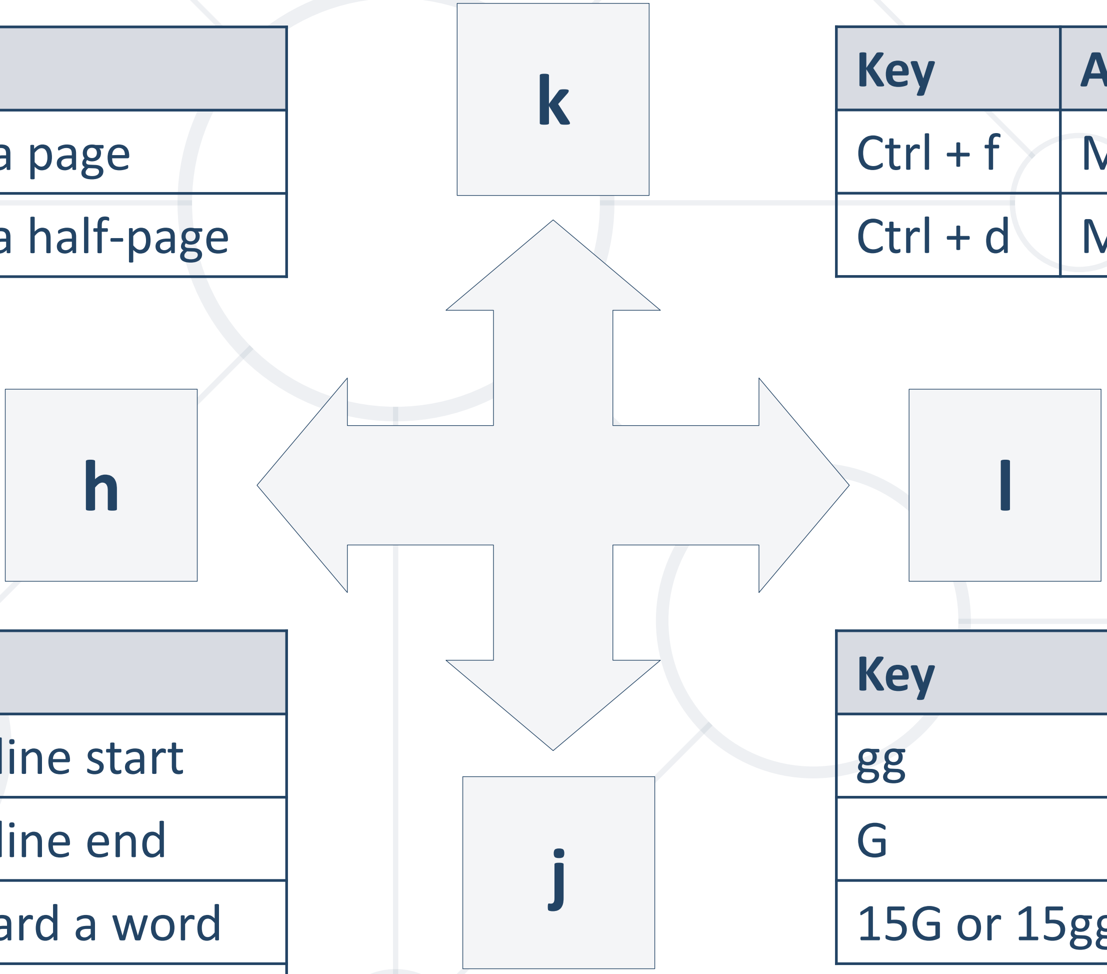
VIM Modes

- Normal (command)
  - Navigation
- Command (ex command or last line mode)
  - Commands are entered after the : symbol
- Insert
- Replace
- Visual

Navigation in Normal Mode

- Using keys **h**, **j**, **k**, and **l**

Key	Action
Ctrl + b	Moves backward a page
Ctrl + u	Moves backward a half-page



k

h

l

j

Key	Action
Ctrl + f	Moves forward a page
Ctrl + d	Moves forward a half-page

Key	Action
^ (Shift + 6)	Moves to the line start
$ (Shift + 4)	Moves to the line end
b	Moves backward a word
w	Moves forward a word

Key	Action
gg	Moves to the first line
G	Moves to the last line
15G or 15gg	Goes to line 15

Enter in Insert Mode*

- **i** key - Insert **here**
- **I** key (Shift + i) - Insert at the **beginning of the line**
- **a** key - Append after **current position**
- **A** key (Shift + a) - Append to the **end of the line**
- **o** key - Open a new **line bellow**
- **O** key (Shift + o) - Open a new **line above**

* From Normal mode

Enter in Replace and Visual Mode*

- **r** key
  - Replace **one symbol under** the cursor
- **R** key (Shift + r)
  - Enters in **replace mode**
- **v** key
  - Enters in visual mode with **custom selection allowed**
- **V** key (Shift + v)
  - Enters in visual mode with **line selection enabled**

* From Normal mode

Deleting Text and Lines*

Key	Delete action
x	Single character under the cursor
X	Single character before the cursor
dw	To the end of a single word under the cursor
3dw	Three words
dd	Current line
d^ or d0	All text from the beginning of the line to the cursor
D or d$	All text from cursor position to the end of the line
dL	All text from the cursor to the end of the screen
dG	All text from the cursor to the end of the document

* Partial list with a few examples

Copy, Paste, and Join*

Key	Action
yy	Copies a line of text
3yy	Copies three lines of text
yw	Copies from the cursor to the end of the word
3yw	Copies three words
p	Pastes to the right of the cursor
P (Shift + p)	Pastes to the left of the cursor
J (Shift + j)	Joins current line to the previous

* Partial list with a few examples

Searching and Replacing

- **Search only**
  - Forward **/string** and backward **?string**
  - Move between occurrences **n** (same direction) and **N** (opposite)
- **Search-and-Replace Syntax**
  - action/string-to-find/replace-with/modifier
  - First instance on the current line - **:s/tcp/TCP/**
  - All instances on the current line - **:s/tcp/TCP/g**
  - All instances - **:%s/tcp/TCP/g**

Undo Changes

- **u** key
  - Undo one change
- **:e!**
  - Re-read the file, discarding all changes

Save Changes

- **:w**
  - Save the file
- **:wq**
  - Save the file and quit
- **:x**
  - Save the file and quit
- **ZZ** (Shift + z + z)
  - Save the file and quit

Quit Commands

- :q
  - Quit if no changes are made without save
- :q!
  - Quit without save

A Few More Scenarios

- **:w another-file.txt**
  - **Save** the file as **another-file.txt**
- **:20,30w /tmp/file.txt**
  - Save the **lines between 20 and 30** to **/tmp/file.txt**
- **:r another-file.txt**
  - Insert the **contents** of **another-file.txt** at the cursor position
- **:r ! uname -a**
  - Insert the **result** from the **uname -a** command at the cursor

VIM Options*

- Set an option
  - **:set number** - turn on the line numbering
  - **:set nonumber** - turn off the line numbering
- List options
  - All options **:set all** or for the current user **:set**
- Store options in a configuration file
  - System level **/etc/virc** or **/etc/vimrc**
  - On user level **~/.vimrc**

* Partial list

Screen Editors

nano

nano *

- Easier for most newcomers
- Offers menu-like navigation
- Most commands are available as key combinations
- Usually, **Ctrl** (displayed as **^**) and **Alt** (displayed as **M**) are used
- Should you need help, you can always press **Ctrl+G**

* Depending on the distribution and the installation type, additional steps may be required.

[~]$ sed

Stream Editors

Characteristics. sed

Stream Editors

- Characteristics
  - Treat the text as stream of characters
  - Can apply transformations on the fly
- Typical stream editors
  - sed, awk

sed

- Description
  - Stream editor for filtering and transforming text
- Example

```
[user@host ~]$ echo 'one twenty-one' | sed s/one/ONE/g
ONE twenty-ONE
...

[user@host ~]$ sed s/one/ONE/g filein.txt > fileout.txt
...
```

Common Sed Scenarios #1

- Replace **first instance**

```
[user@host ~]$ sed s/tcp/TCP/ file.txt
```

- Replace **all instances**

```
[user@host ~]$ sed s/tcp/TCP/g file.txt
```

- Two consecutive search and replace operations

```
[user@host ~]$ sed 's/tcp/TCP/g ; s/TCP/UDP/g' file.txt
...
[user@host ~]$ sed -e s/tcp/TCP/g -e s/TCP/UDP/g file.txt
...
```

Common Sed Scenarios #2

- Replace pattern with spaces

```
[user@host ~]$ sed 's/is not/is too/g' file.txt
```

- Replace all instances, but print only the changed ones

```
[user@host ~]$ sed -n s/dns/DNS/pg /etc/services
```

- Search and replace in rage of lines

```
[user@host ~]$ sed -n '1,10s/dns/DNS/pg' services
```

- Delete comment and empty lines and create a backup

```
[user@host ~]$ sed –i.bak '/^#/d;/^$/d' services
```

[~]$ awk

Stream Editors

awk

awk

- Each line of text is a **record**
- Lines are separated based on the **carriage return/line feed** char
- Every record can have **different amount of fields**
- Each word in the line, separated by a space or tab is a **field**
- Fields are referenced by ***$numbers***
- The first field is **$1**, second is **$2** and so on

awk

- Description
  - Pattern scanning and processing language
- Example

```
# print the first two fields of every line
[user@host ~]$ cat file.txt | awk '{print $1,$2}'

# using different field separator
[user@host ~]$ cat /etc/passwd | awk -F ':' '{print $1,$7}'

# use only lines containing the word text
[user@host ~]$ cat file.txt | awk '/text/ {print $1,$7}'
```

Other Use Cases

Other use Cases of Vim

vipw

- Description
  - Edit the passwd or shadow-password file
- Example

```
[root@host ~]# vipw
...
user:x:1000:1000::/home/user:/bin/bash
devops:x:1001:1001::/home/devops:/bin/bash
clerk:x:1002:1002::/home/clerk:/bin/bash
...
```

vigr

- Description
  - Edit the group or shadow-group file
- Example

```
[root@host ~]# vigr
...
user:x:1000:user
devops:x:1001:devops
clerk:x:1002:clerk
...
```

visudo

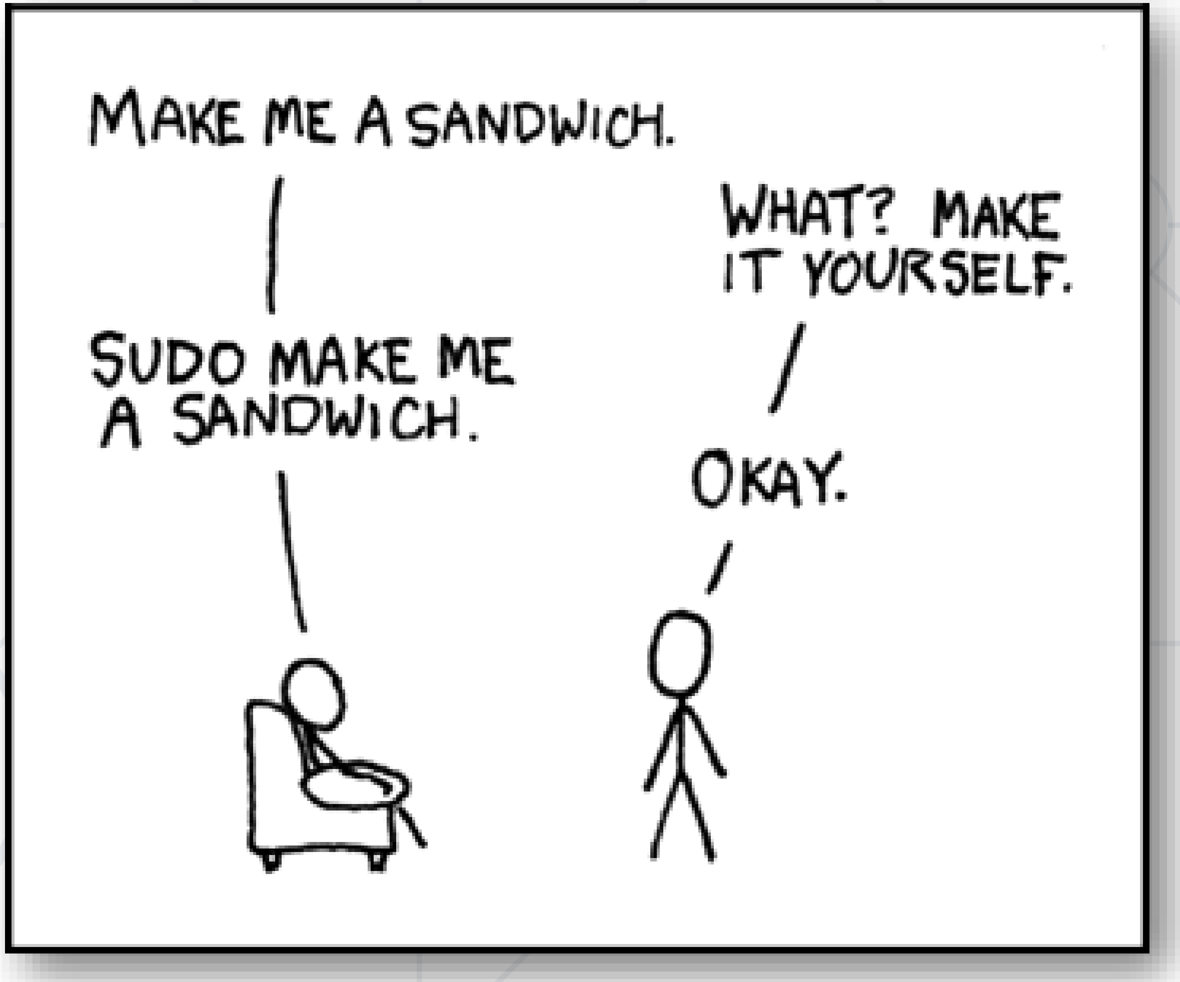
- Description
  - Edit the sudoers file
- Example

```
[root@host ~]# visudo
...
# Allow root to run any commands anywhere
root    ALL=(ALL)    ALL
...
```

SUDO Management

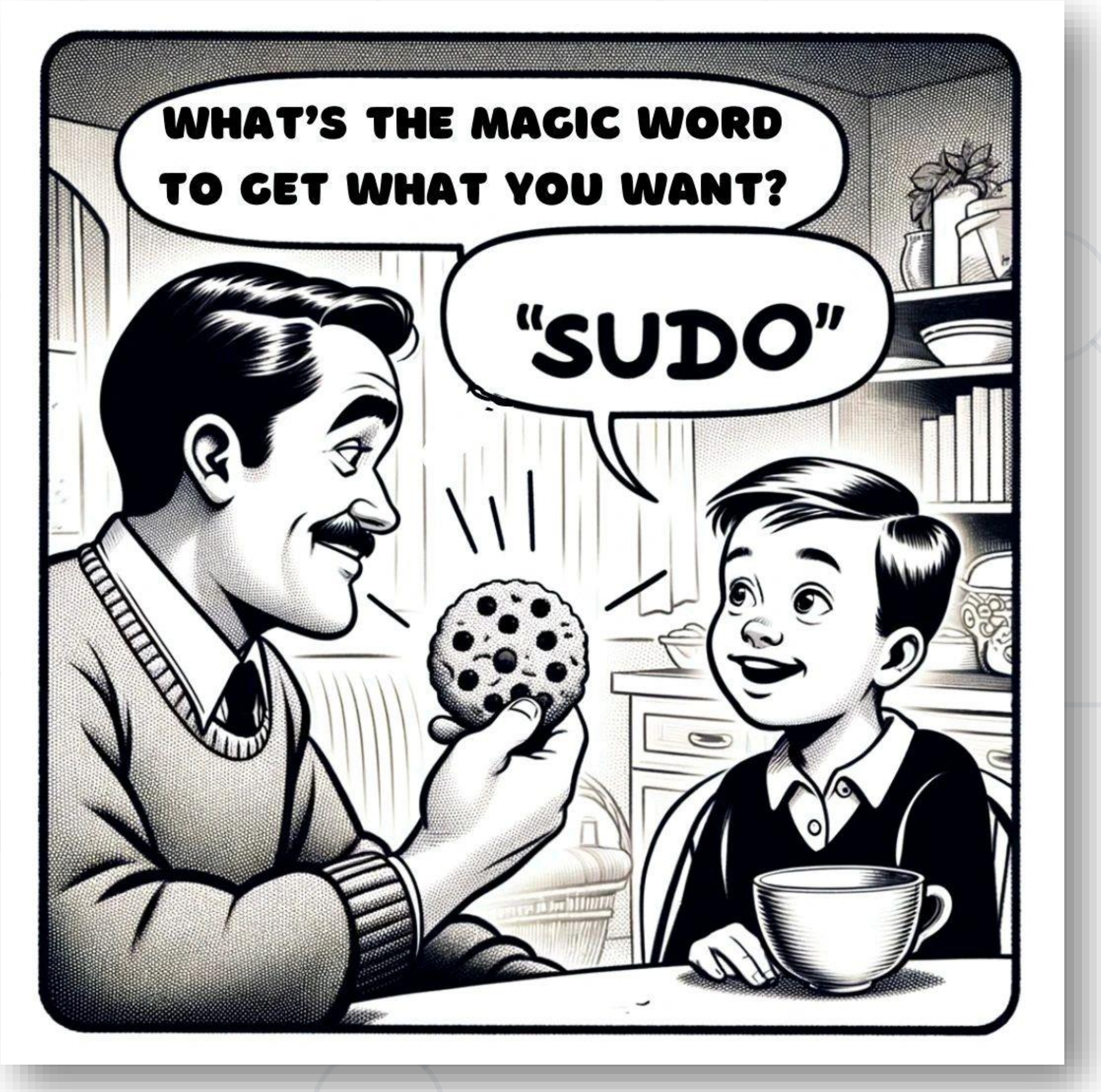
Other means of controlling SUDO

The Sandwich Request ☺



https://xkcd.com/149/

The Magic Word



https://www.linkedin.com/posts/atomycloud_whats-the-magic-word-to-get-what-you-activity-7159299509129228288-MsZF/

SUDO Configuration

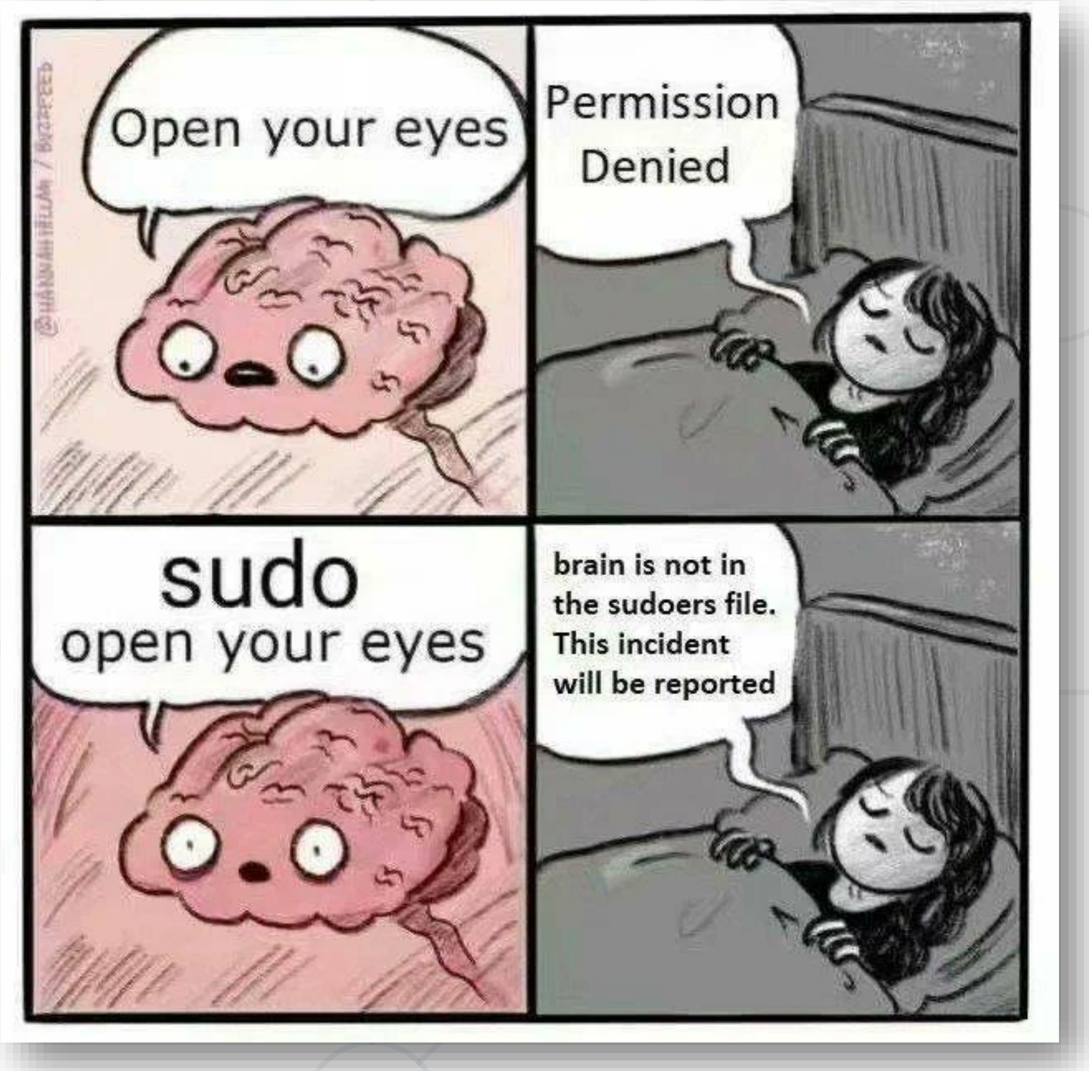
- Control **who** can do **what** and from **where**
- Main configuration file
  - **`/etc/sudoers`**
- Additional configuration files *(same structure, no extensions)*
  - **`/etc/sudoers.d/`**
- Can be managed with group membership as well
  - For **Red Hat** and **openSUSE** families – **wheel**
  - For **Debian**-based distributions – **admin** or **sudo**
- Supports aliases for **users**, **hosts**, and **commands**

SUDO File Format

- Main configuration instructions
  - For user: **user (host)=(user:group) [options] commands**
  - For group: **%group (host)=(user:group) [options] commands**
- Examples

```
# root can execute any command as anyone from anywhere
root       ALL=(ALL:ALL)          ALL
# (shorter alternative of the above)
root       ALL=(ALL)              ALL
# members of group can execute any command as anyone from anywhere
%wheel     ALL=(ALL)              ALL
# user can execute any command as anyone from anywhere w/o password
demo       ALL=(ALL)    NOPASSWD: ALL
# user can execute specific command as anyone from anywhere w/o password
demo       ALL=(ALL)    NOPASSWD: /usr/bin/command
```

Magic Does Not Always Work As Expected



https://x.com/bearstech/status/1689212422693761024

Practice

Summary

- stdin, stdout, and stderr are the three system streams or descriptors
- They can be redirected with operators like <, > , <<, and >>
- Multiple redirection instructions are read fro m left to right
- We can create command sequences with the help of ;, |, &&, ||
- Commands in the sequences can be (in)dependent on each other

Summary

- We can link the output of one command to the next, and then to another, …
- wc, cut, tac, head, tail, and sort are just several of the text processing tools
- grep and find allow us to search in or for files
- Vim is integral part of our toolset. It is very powerful and minimalistic editor
- However, sed and awk cover tasks that require stream editing

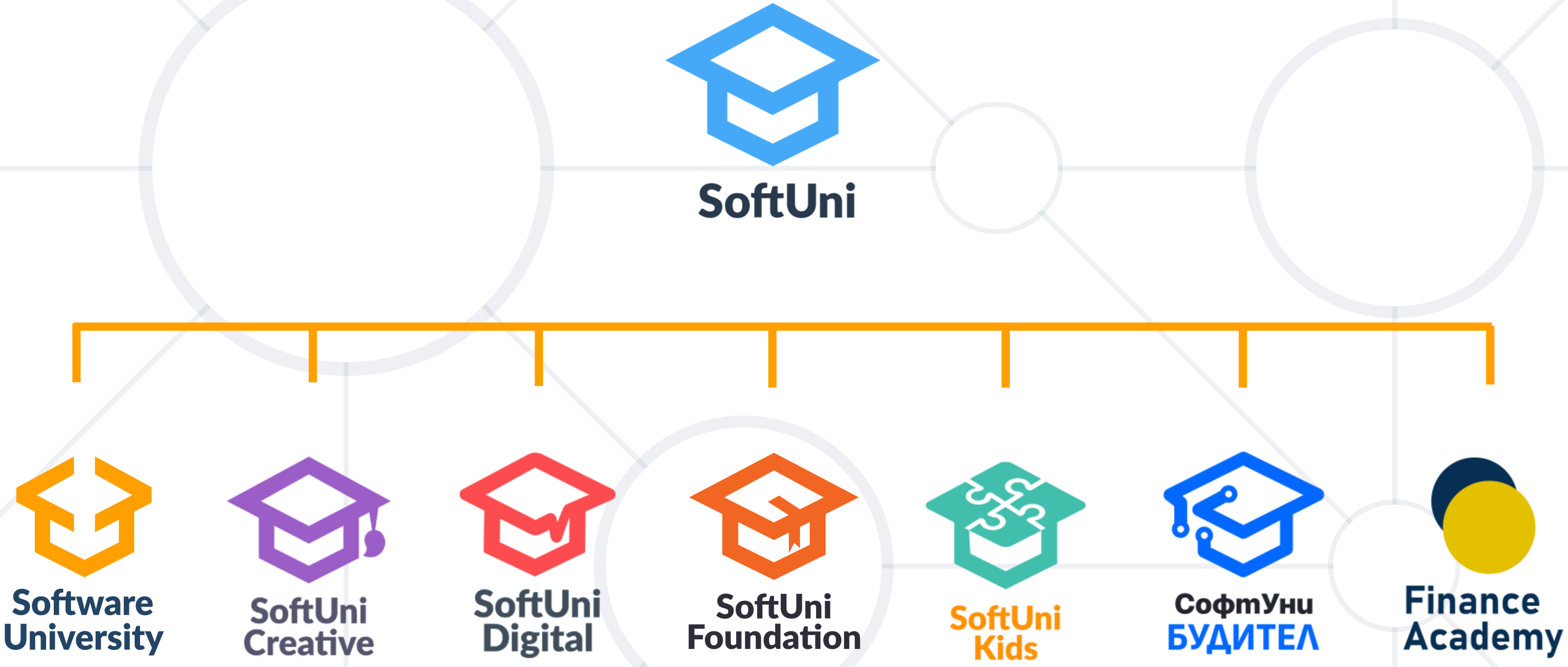
Resources

- The Linux Command Line
  - http://linuxcommand.org/tlcl.php
- Bash Guide for Beginners
  - http://tldp.org/LDP/Bash-Beginners-Guide/html/index.html
- Bash Reference Manual
  - https://www.gnu.org/software/bash/manual/html_node/index.html

Resources

- Vim Home
  - [*http://www.vim.org/*](http://www.vim.org/)
- Vim Adventures
  - [*https://vim-adventures.com/*](https://vim-adventures.com/)
- sed Manual
  - [*https://www.gnu.org/software/sed/manual/sed.html*](https://www.gnu.org/software/sed/manual/sed.html)

Questions?



SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

СофтУни БУДИТЕЛ

Finance Academy

SoftUni Diamond Partners

Software University

SmartIT

Coca-Cola HBC Bulgaria

SUPER HOSTING .BG

createX

Postbank

top EMPLOYER България Bulgaria 2024 FOR A BETTER WORLD OF WORK

DRAFT KINGS

THE CROWN IS YOURS

VIVACOM

INDEAVR

Serving the high achievers

AMBITIONED

PHAR VISION

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity

License