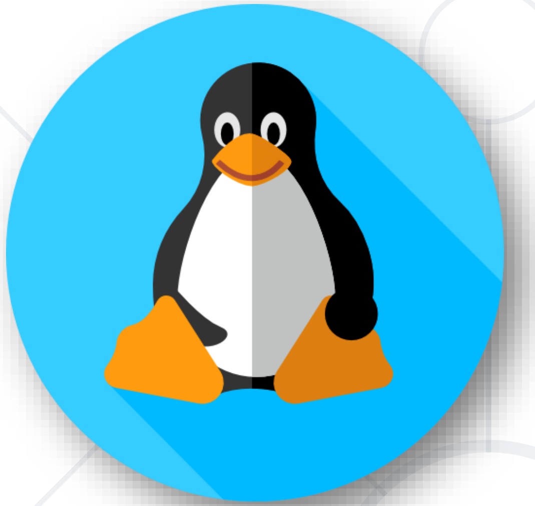
Working in the Console

Getting Help. Working with Files and Folders

Users and Groups. Access Rights



SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

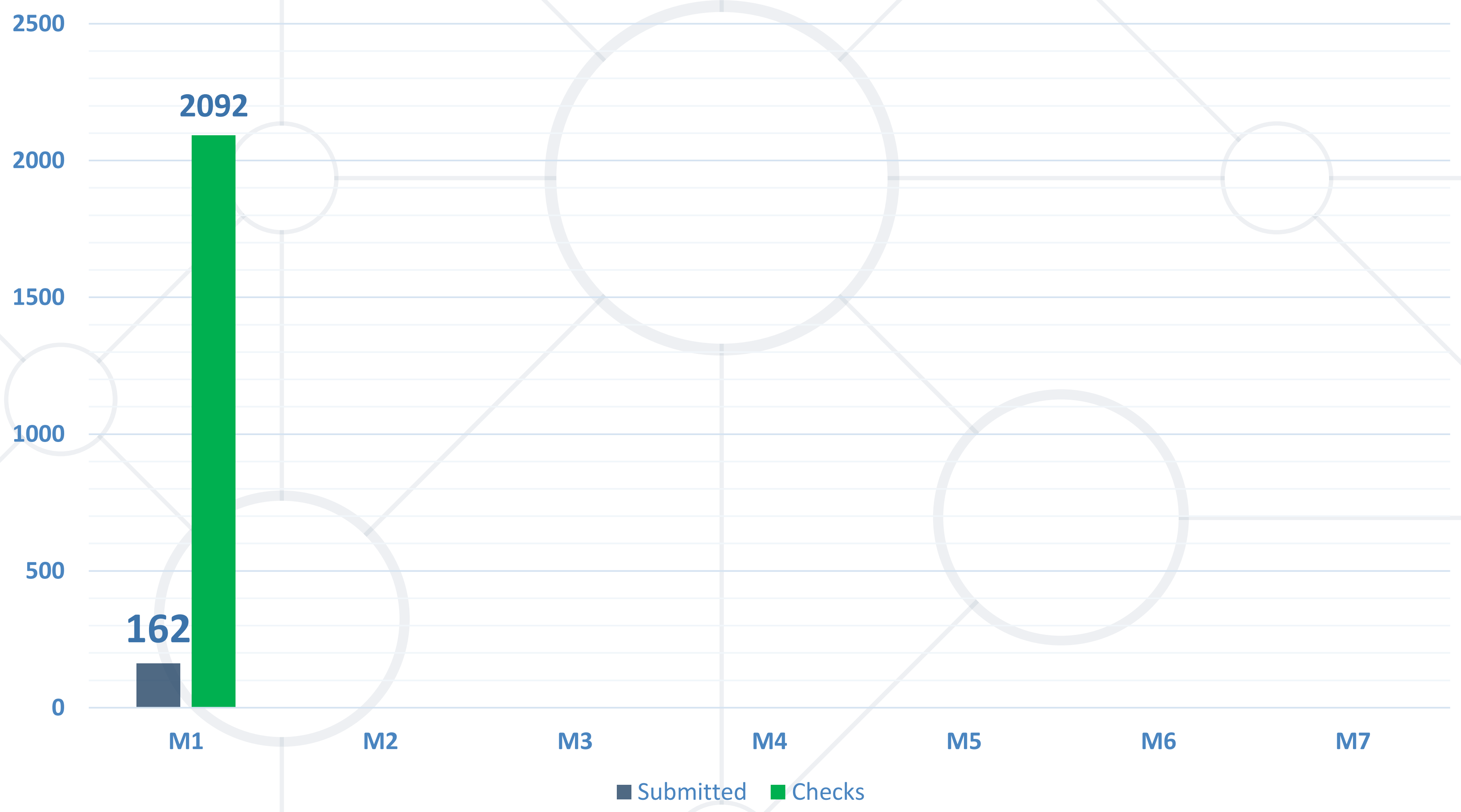
https://softuni.bg

Have a Question?

sli.do

#LSA

Homework Progress (as of 14:30)



Solutions for **M1** can be submitted until **23:59:59** on **13.03.2025**

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

Quick Overview

Previous Module (M1)

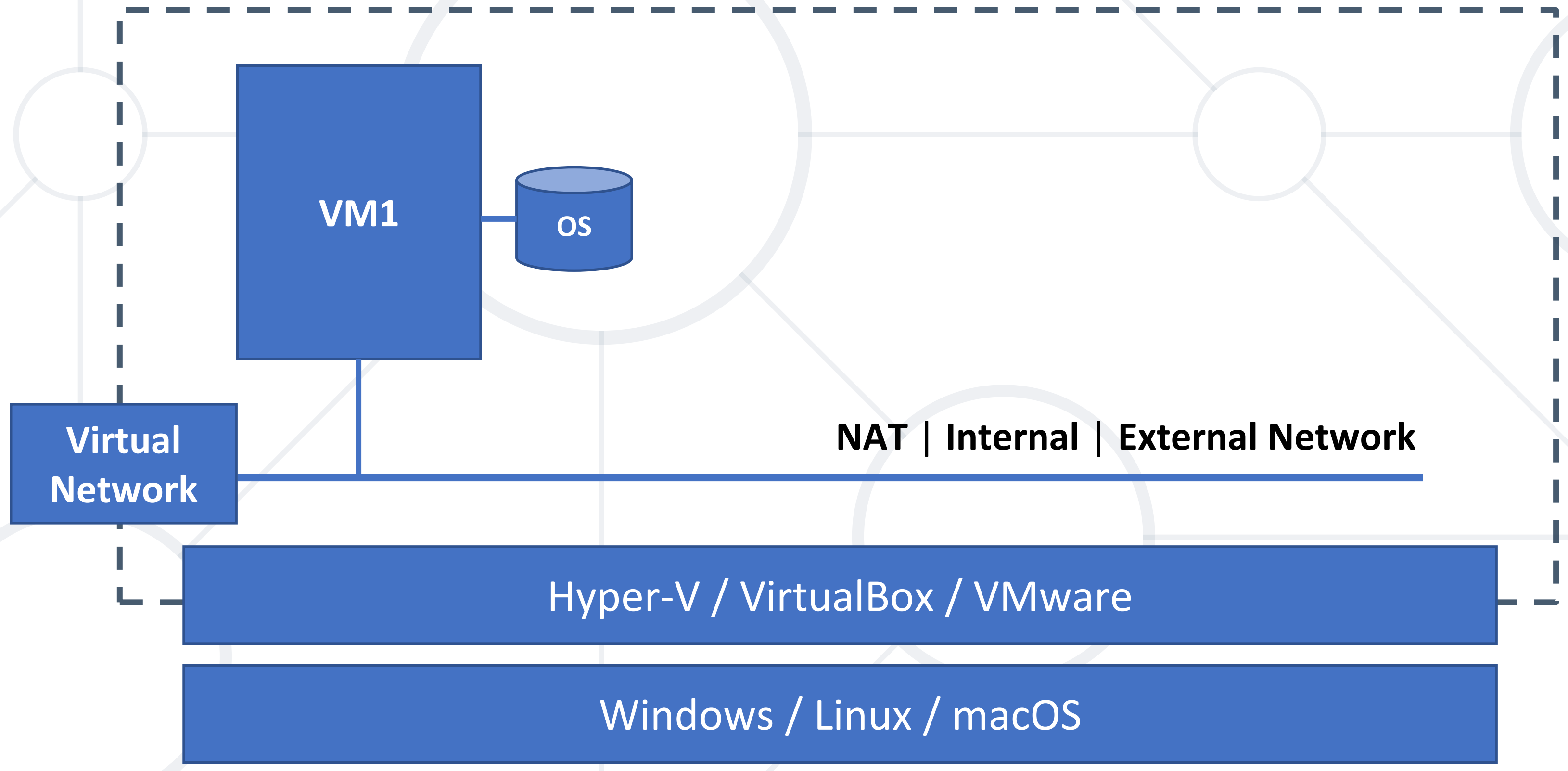
Table of Contents

This Module (M2)

Topics and Lab Infrastructure

Table of Contents

Lab Infrastructure


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

Console Deep Dive

Environmental: Definitions and Tools

Environment

- Sets the **operational conditions**
- Driven by **variables** (PATH, USER, SHELL, …)
- Set up both on **system level** and **per user**
- It is **modifiable** by both users and processes
- It is **inheritable**

Special Environment Variables

- General purpose
  - **$?** => Return the exit code of last executed command
  - **$!** => Return the PID of the last job run in background
  - **$$** => Return the PID of the current process
  - **$_** => Return the final argument of the previous command
- Prompt related
  - **$PS1** => Regular prompt
  - **$PS2** => Prompt during multi-line commands

Prompt Macros

Code	Display
\h	Hostname until the first '.'
\H	Full hostname
\t	Current time in 24-hour format HH:MM:SS
\A	Current time in 24-hour format HH:MM
\u	Username of the current user
\w	Current working directory
\W	Base name of the current working directory
\#	Command number of this command
\$	If UID=0 then it is '#' otherwise it is '$'

set

- Purpose
  - Controls shell options. Display values of shell variables
- Syntax

```
set [options] [+/-o shell options] [arguments]
```

- Examples

```
# Display shell options suitable for re-use
[user@host ~]$ set +o
# Display all shell variable names and values
[user@host ~]$ set
```

unset

- Purpose
  - Unset values and attributes of shell variables and functions
- Syntax

```
unset [options] [name]
```

- Examples

```
# Unset single variable
[user@host ~]$ unset MYVAR1
# Unset multiple variables
[user@host ~]$ unset -v MYVAR1 MYVAR2 MYVAR3
```

Command Execution

Executable Artifacts. Order of Execution

Executable Artifacts

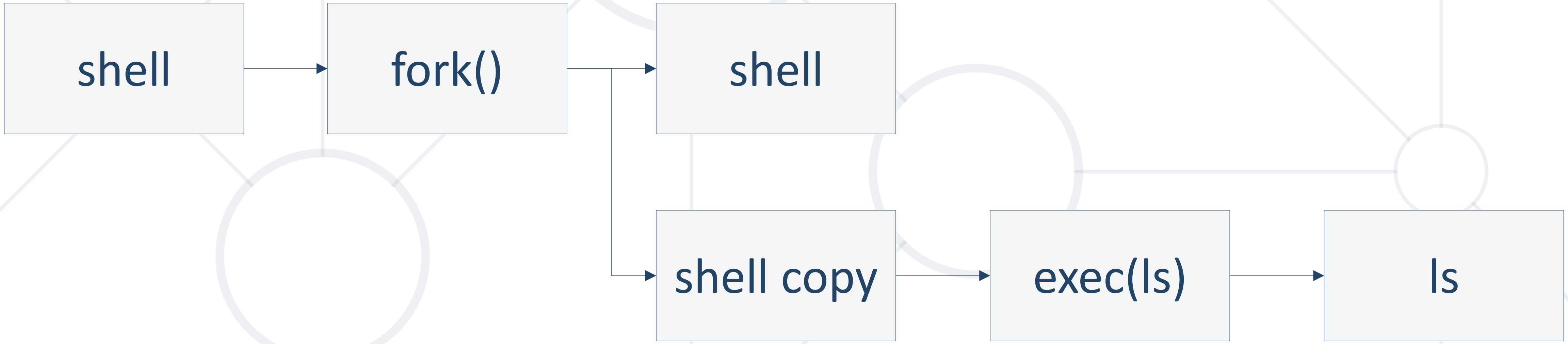
- Shell Built-in Commands
- External Commands
  - Scripts
  - Binary Files
- Special Types
  - Aliases
  - Functions

Command Execution (Shell's Perspective)

- When we execute this

```
[user@host ~]$ ls
```

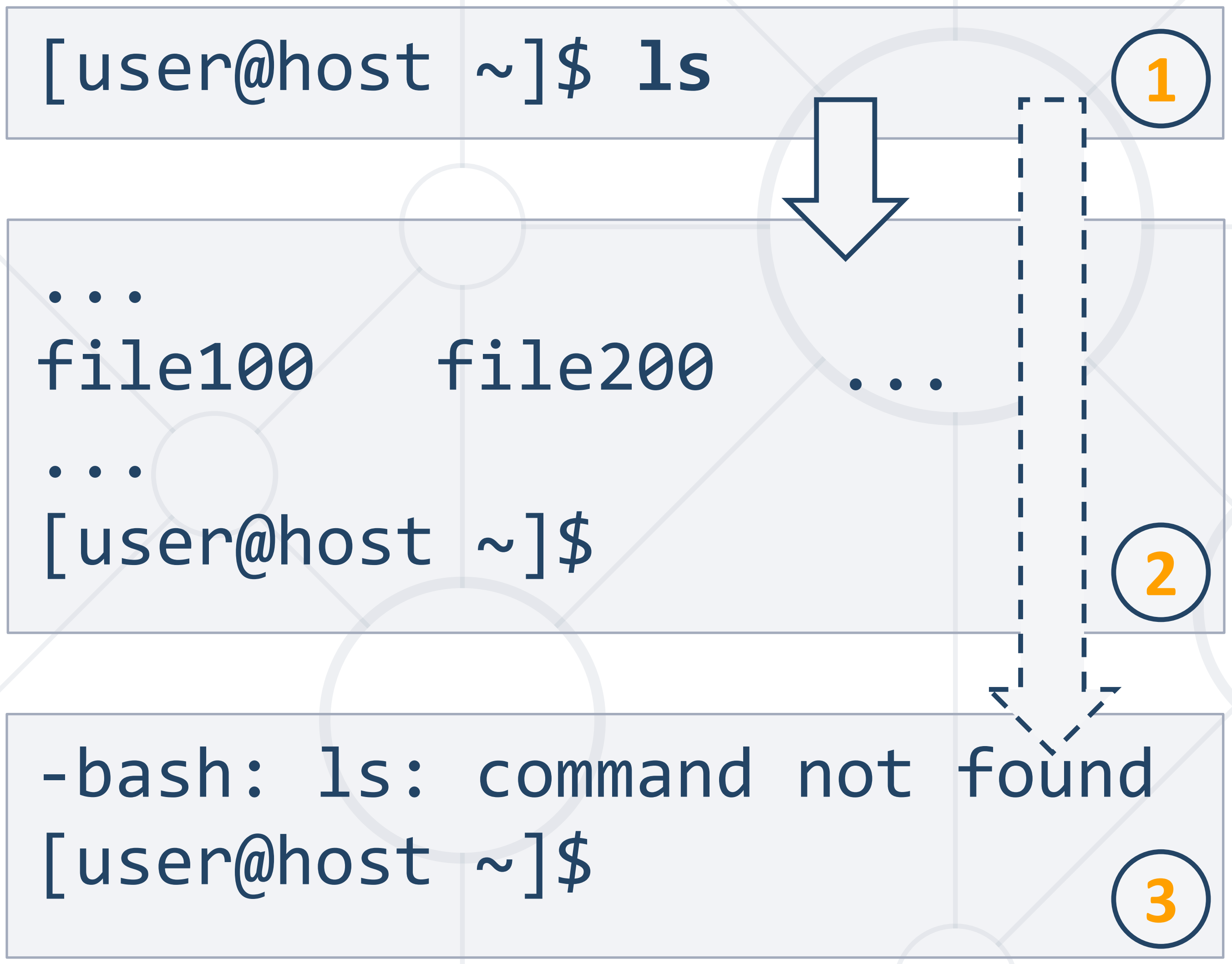
- This is what happens



Sourcing vs. Execution

- **Sourcing**
  - No subshell is created
  - Any variables set become part of the environment
  - Methods: `. script.sh` or `source script.sh`
- **Execution**
  - Subshell is **always** created (except for the built-in commands)
  - No subshell if using `exec ./script.sh`

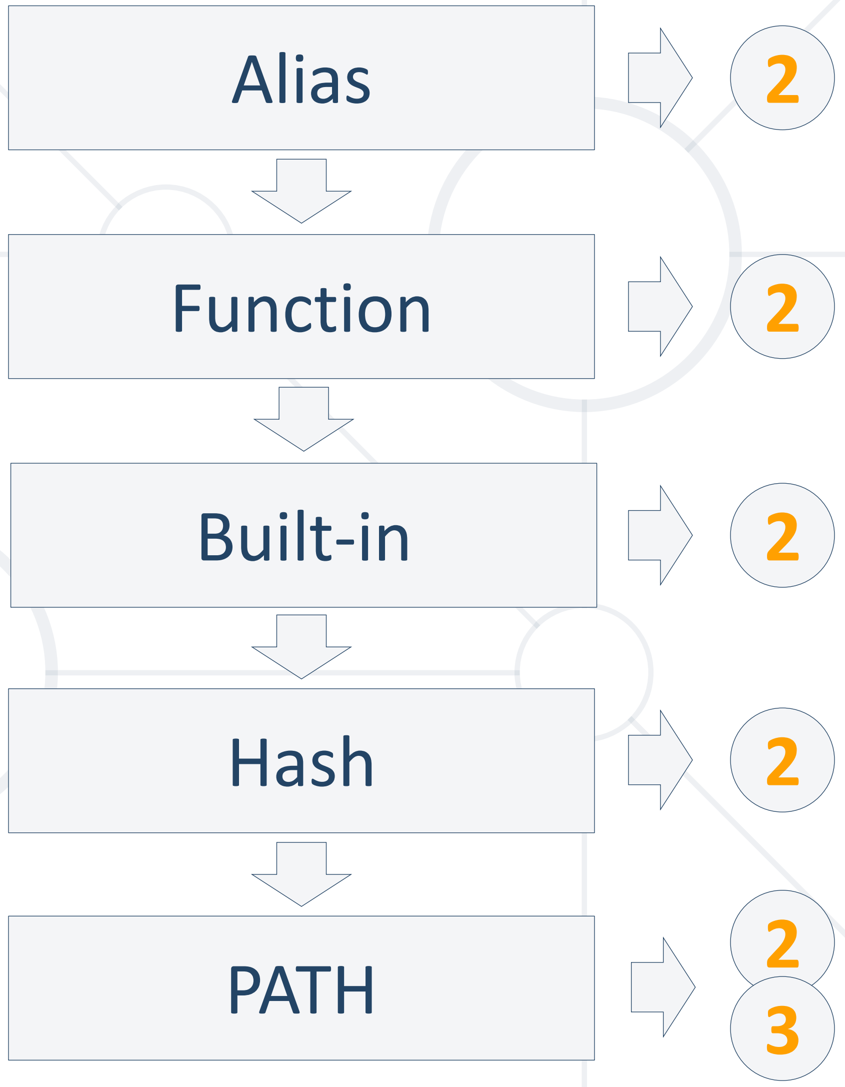
Execution (Search) Order



```
[user@host ~]$ ls
```

```
...
file100   file200   ...
...
[user@host ~]$
```

```
-bash: ls: command not found
[user@host ~]$
```



Break the Order

- Force Built-in Usage

```
[user@host ~]$ builtin test
```

- Set Explicit Path

```
[user@host ~]$ /bin/test
```

- Ignore Aliases and Functions

```
[user@host ~]$ command test
```

- Ignore Just Aliases

```
[user@host ~]$ \test
```

hash

- Purpose
  - Remembers or display program locations
- Syntax

```
hash [options] [name]
```

- Examples

```
# Display re-usable list of program locations
[user@host ~]$ hash -l
# Add a program location to the list
[user@host ~]$ hash -p /bin/ping ping
```

whereis

- Purpose
  - Locates the binary, source, and man page files for a command
- Syntax

```
whereis [options] name [name …]
```

- Examples

```
# Display all files for a command
[user@host ~]$ whereis ls
# Display only binary file information
[user@host ~]$ whereis -b ls
```

which

- Purpose
  - Shows the full path of (shell) commands
- Syntax

```
which [options] name [name …]
```

- Examples

```
# Show what would have been executed
[user@host ~]$ which cd
# Print all matching executables in PATH
[user@host ~]$ which -a cd
```

type

- Purpose
  - Displays information about command type
- Syntax

```
type [options] name [name …]
```

- Examples

```
# Show everything about a single command
[user@host ~]$ type -a ls
# Print information about multiple commands
[user@host ~]$ type cd ls pwd
```

alias

- Purpose
  - Define or display aliases
- Syntax

```
alias [-p] [name[=value]]
```

- Examples

```
# Print all aliases in re-usable format
[user@host ~]$ alias -p
# Define new alias
[user@host ~]$ alias si='uname -a'
```

unalias

- Purpose
  - Removes alias
- Syntax

```
unalias [-a] name [name …]
```

- Examples

```
# Remove all aliases
[user@host ~]$ unalias -a
# Remove two aliases
[user@host ~]$ unalias ls ll
```

export

- Description
  - Sets export attribute for shell variables
- Example

```
[user@host ~]$ export MYVAR=100
[user@host ~]$ bash
[user@host ~]$ echo $MYVAR
100
[user@host ~]$ MYVAR=200
[user@host ~]$ echo $MYVAR
200
[user@host ~]$ exit
[user@host ~]$ echo $MYVAR
100
```

Child (2-nd) shell

env

- Description
  - Runs a program in a modified environment
- Example

```
[user@host ~]$ env MYVAR=100 PS1="CHILD:$PS1" bash
CHILD:[user@host ~]$ echo $MYVAR
100
CHILD:[user@host ~]$ MYVAR=200
CHILD:[user@host ~]$ echo $MYVAR
200
CHILD:[user@host ~]$ exit
[user@host ~]$ echo $MYVAR
```

Child (2-nd) shell

Configuration Files

What Drives the BASH Shell?

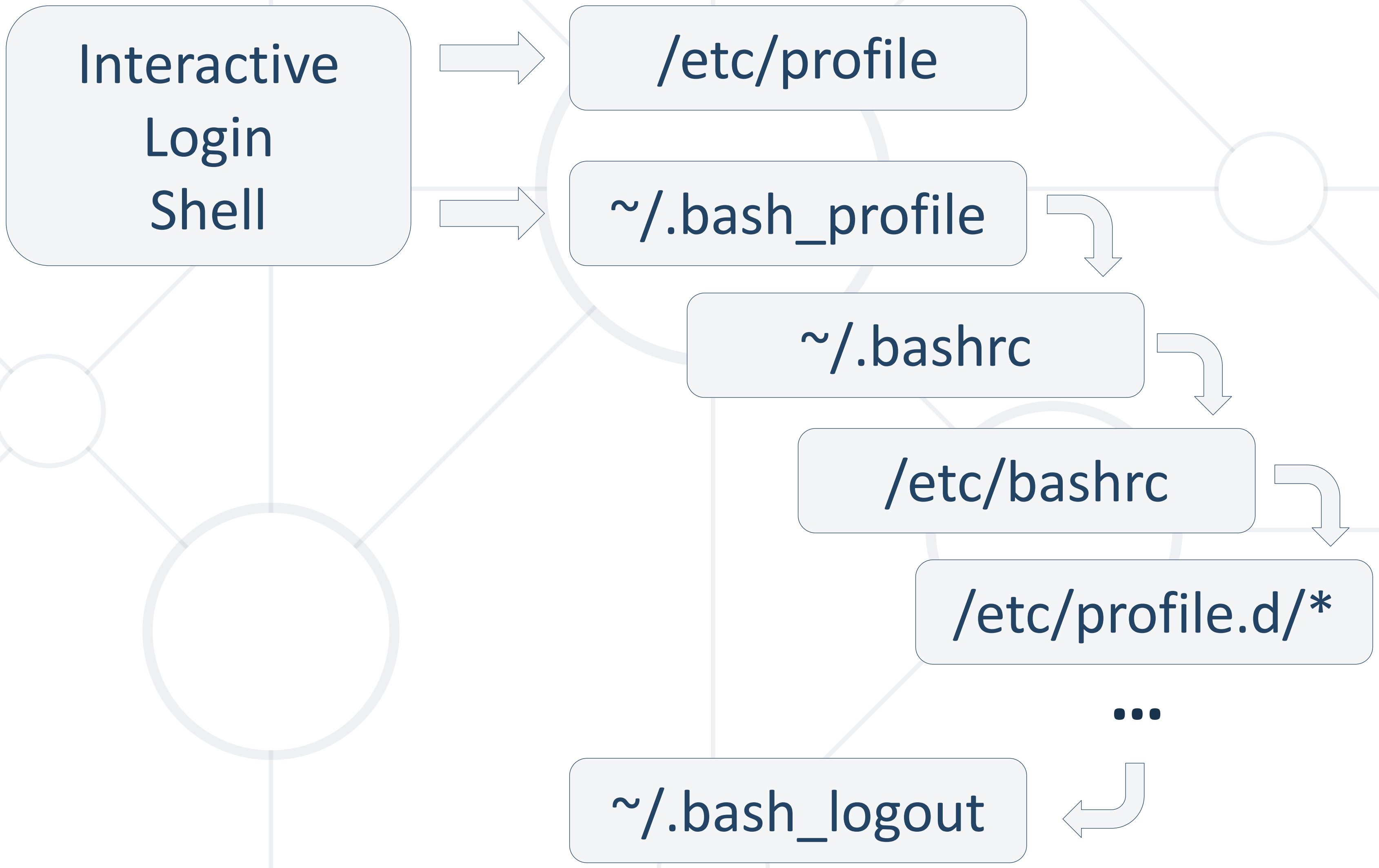
System Level Configuration

- **Stored** in **/etc**
- **File profile**
  - Used for **Environment** control and **Startup** programs execution
- **File bashrc** (Red Hat) or **bash.bashrc** (Debian, openSUSE)
  - Used for **Functions** and **Aliases** definition
- **Folder profile.d/***
  - Used for custom routines definition
  - It is read by **profile** (all), **bashrc** (Red Hat), and **bash.bashrc** (openSUSE)

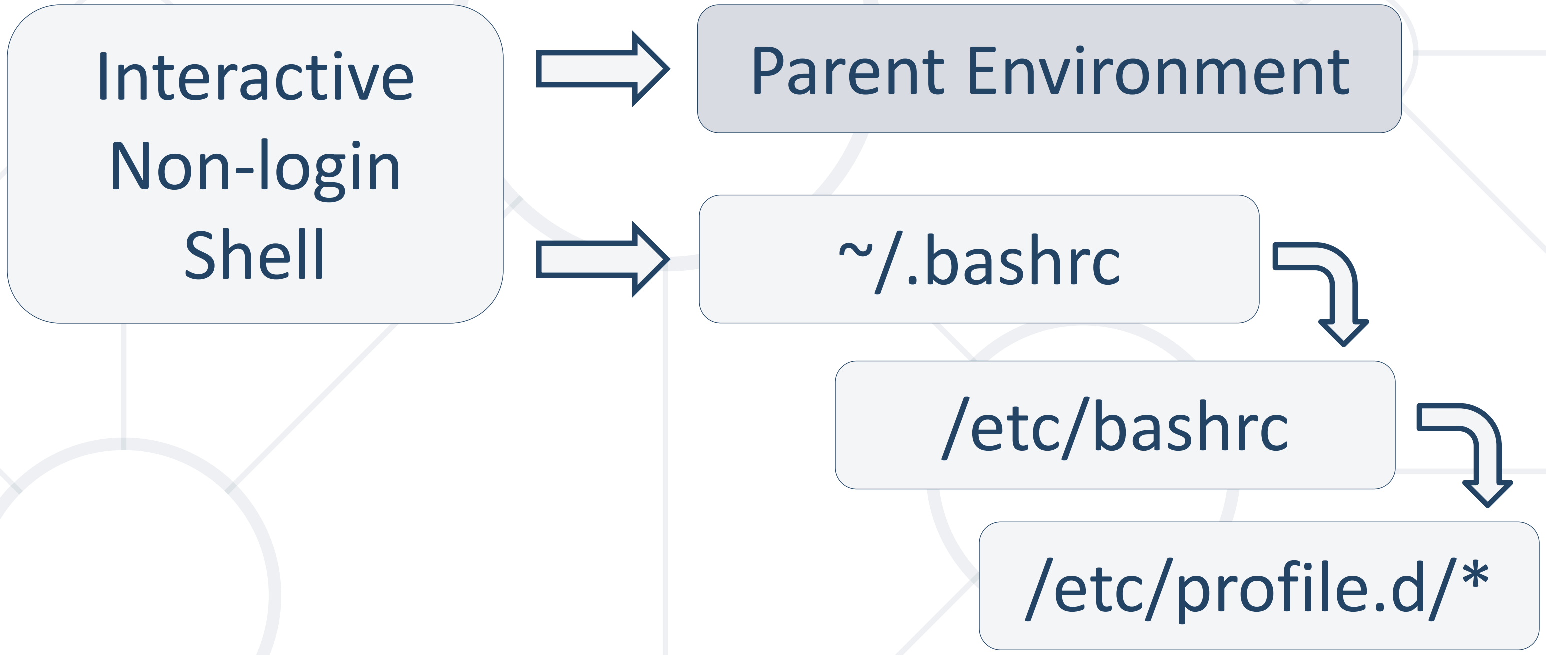
User Level Configuration

- **Stored** in **user's home directory**
- **File .bash_profile** (Red Hat) or **.profile** (Debian, openSUSE)
  - Executed only in **login shell**
  - Reads **~/.bashrc**
- **File .bashrc** (all)
  - Executed **always**
  - Reads **/etc/bashrc** (Red Hat)

Login Shell Sequence



Non-login Shell Sequence



Practice

Getting Help

Know Them. Use Them

Many Help Sources

- Installed locally
  - Internal
  - External
- On-line
  - Forums
  - Community
  - Mail lists

On-premise. Level of detail varies. Comes directly from the authors

Real life experience. Usually takes time to filter and find the right answer

--help (-h or -?)

- Description
  - Display short usage information about a command
- Example

```
[user@host ~]$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILEs
...
  -a, --all   do not ignore entries starting with .
...
```

help

- Purpose
  - Display information about the built-in commands
- Syntax

```
help [options] [command]
```

- Examples

```
# Show all built-in commands
[user@host ~]$ help
# Print information about a command
[user@host ~]$ help type
```

man

- Purpose
  - The system's manual pager
- Syntax

```
man [options] [page]
```

- Examples

```
# Show information about man itself
[user@host ~]$ man man
# Show section of a page
[user@host ~]$ man 5 passwd
```

Sections in man (Page)

- NAME
- SYNOPSIS
- CONFIGURATION
- DESCRIPTION
- OPTIONS
- EXIT STATUS
- RETURN VALUE
- ERRORS
- ENVIRONMENT
- FILES
- VERSIONS
- CONFORMING TO
- NOTES
- BUGS
- EXAMPLE
- AUTHORS

Categories (Sections) in man

- Executable programs (1)
- System calls (2)
- Library calls (3)
- Special files (4)
- File formats and conventions (5)
- Games (6)
- Miscellaneous (7)
- System administration commands (8)
- Kernel routines (9)

whatis (man -f)

- Purpose
  - Displays manual page description
- Syntax

```
whatis [options] name [name …]
```

- Examples

```
# Display information about ls
[user@host ~]$ whatis ls
# Display information about multiple commands
[user@host ~]$ whatis pwd uname alias
```

apropos (man -k)

- Purpose
  - Search the manual page names and descriptions
- Syntax

```
apropos [options] keyword
```

- Examples

```
# Show information about command
[user@host ~]$ apropos passwd
# Show information when all keywords match
[user@host ~]$ apropos -a passwd user
```

info

- Purpose
  - Read Info documents
- Syntax

```
info [options] [menu-term]
```

- Examples

```
# Open top-level menu
[user@host ~]$ info
# Start at the beginning of the page for a program
[user@host ~]$ info passwd
```

Accompanying Documentation

- Description
  - Documentation included with the installed software
- Path
  - **`/usr/share/doc`**
- Usage

```
[user@host ~]$ cat /usr/share/doc/system/README
systemd System and Service Manager
...
```

Working with Files

Explore, Know, and Rule Those Files

Everything is Files

- **Naming conventions**
  - Case **sensitive** (file.txt <> File.txt <> FILE.TXT)
  - Stick to **alphanumeric** characters
  - Substitute **spaces** with (**_** , **-** , **.**)
  - **Extensions** are not needed, but **nice to have**
- **Work with multiple files**
  - Helper symbols when reading or listing (*, ? , [] , {})
  - Techniques when creating ({X,Y,Z}, {A..D}, {1..10})

File Types

- Regular (-)
- Directory (**d**)
- Symbolic link (**l**)
- Block device (**b**)
- Character device (**c**)
- Named pipe (**p**)
- Socket (**s**)

Regular files

Special files

First Symbol in the Long Listing

file

- Purpose
  - Determine file type
- Syntax

```
file [options] file [file …]
```

- Examples

```
# Show information about a file
[user@host ~]$ file /etc/profile
# Show information about multiple files
[user@host ~]$ file /etc/*.conf
```

stat

- Purpose
  - Display file or file system status
- Syntax

```
stat [options] file [file …]
```

- Examples

```
# Show information about a file
[user@host ~]$ stat .bash_history
# Show information about files in a special format
[user@host ~]$ stat --terse /etc/*.conf
```

touch

- Purpose
  - Change file timestamp
- Syntax

```
touch [options] file [file …]
```

- Examples

```
# Change access time of a file
[user@host ~]$ touch -a .bash_history
# Create an empty file
[user@host ~]$ touch emptyfile.txt
```

cp

- Purpose
  - Copy files and directories
- Syntax

```
cp [options] source dest
```

- Examples

```
# Copy single file
[user@host ~]$ cp file1.txt ~/Documents/my-file.txt
# Copy multiple files to a folder
[user@host ~]$ cp /etc/*.conf ~/Temp/
```

mv

- Purpose
  - Move (rename) files
- Syntax

```
mv [options] source dest
```

- Examples

```
# Rename a file
[user@host ~]$ mv fileA.txt fileB.txt
# Move multiple files to a folder
[user@host ~]$ mv *.bak ~/Backup/
```

rm

- Purpose
  - Remove files or directories
- Syntax

```
rm [options] file [file …]
```

- Examples

```
# Remove multiple files
[user@host ~]$ rm file?.txt
# Remove folder and its contents
[user@host ~]$ rm -rf ~/Temp
```

mkdir

- Purpose
  - Make directories
- Syntax

```
mkdir [options] directory [directory …]
```

- Examples

```
# Create two directories
[user@host ~]$ mkdir dir1 dir2
# Create nested directories
[user@host ~]$ mkdir -pv projects/project{1..5}
```

rmdir

- Purpose
  - Remove empty directories
- Syntax

```
rmdir [options] directory [directory …]
```

- Examples

```
# Remove two empty directories
[user@host ~]$ rmdir dir1 dir2
# Remove directory and its ancestors
[user@host ~]$ rmdir -pv projects/project1/phaseA
```

ln

- Purpose
  - Make links between files
- Syntax

```
ln [options] target link_name                    (1st form)
```

- Examples

```
# Create a hard link
[user@host ~]$ ln file.txt ~/Documents/fileH.txt
# Create a soft link
[user@host ~]$ ln -s file.txt ~/Documents/fileS.txt
```

Absolute vs Relative Path

- **Absolute Path** (**starts** with **/**)
  - Calculated from the root of the file system tree
- **Relative Path** (**no leading** **/**)
  - Calculated from the current working directory
- If we are in **/home/user** and we want to create **/shared/temp**

```
# Absolute notation
[user@host ~]$ sudo mkdir -p /shared/temp

# Relative notation
[user@host ~]$ sudo mkdir -p ../../shared/temp
```

Practice

Users and Groups

Manage Users and Groups

Users (Main File)

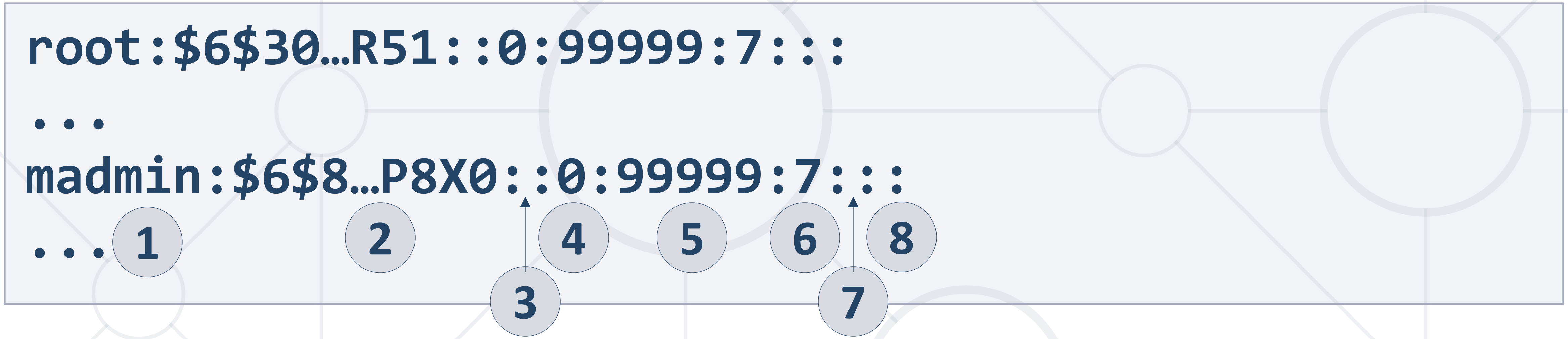
- Users file (/etc/passwd)

```
root:x:0:0:root:/root:/bin/bash
...
madmin:x:1000:1000:M.Admin:/home/madmin:/bin/bash
...  1   2   3    4     5        6            7
```

1	Username (login)
2	Password placeholder
3	User ID
4	Group ID
5	Comment (Full name, phone, etc.)
6	Home directory
7	User shell

Users (Password File)

- Passwords file (**/etc/shadow**)



```
root:$6$30…R51::0:99999:7:::
...
madmin:$6$8…P8X0::0:99999:7:::
...
```

1	Username (login)
2	Encrypted password
3	Last password change
4	Minimum days between change
5	Maximum days validity
6	Warn before expire (days)
7	Inactivity days* after password expire (The number of days after a password has expired during which the password should still be accepted)
8	Account expiration date

* Its meaning could vary amongst UNIX-like OSes. For example, in Solaris it is a little bit different - https://docs.oracle.com/cd/E88353_01/html/E37852/shadow-5.html

User Defaults* During Creation

- Default **values**
  - Read from file **/etc/login.defs**
  - Read from file **/etc/default/useradd**
- Default home **files**
  - Taken (copied) from **/etc/skel/**
  - It could contain both **files** and **directories**

* Default behavior may vary between distributions. For example, not in every distribution, a home folder is being created automatically

Groups (Main File)

- Groups file (/etc/group)

```
root:x:0:
...
wheel:x:10:madmin 4
...
madmin:x:1000:
... 1  2   3
```

1 Group name	3 Group ID
2 Password placeholder	4 Group members

Groups (Password File)

- Groups file (/etc/gshadow)

```
root:::
...
wheel:::madmin  4
...
madmin:!!::madmin
...  1    2  3
```

1 Group name

2 Encrypted password

3 Group administrators

4 Group members

useradd

- Purpose
  - Create a new user or update default new user information
- Syntax

```
useradd [options] login
```

- Examples

```
# Create new user
[user@host ~]$ sudo useradd newuser
# Set a default expiry date
[user@host ~]$ sudo useradd -D -e 2019-12-31
```

usermod

- Purpose
  - Modify a user account
- Syntax

```
usermod [options] login
```

- Examples

```
# Change user's full name (comment field)
[user@host ~]$ sudo usermod -c 'Demo' newuser
# Add user to a group
[user@host ~]$ sudo usermod -aG demogroup newuser
```

userdel

- Purpose
  - Delete a user account and related files
- Syntax

```
userdel [options] login
```

- Examples

```
# Remove a user without removing its home folder
[user@host ~]$ sudo userdel newuser
# Remove a user and its home folder
[user@host ~]$ sudo userdel -r newuser
```

adduser

- Purpose
  - Create a new user (regular or system)*
- Syntax

```
adduser [options] user
```

- Examples

```
# Create new user
[user@host ~]$ sudo adduser helpdesk
# Add an existing user to an existing group
[user@host ~]$ sudo adduser helpdesk itstaff
```

* Reads configuration in /etc/adduser.conf

deluser

- Purpose
  - Remove users (regular or system)*
- Syntax

```
deluser [options] user
```

- Examples

```
# Remove user
[user@host ~]$ sudo deluser helpdesk
# Remove user from a group
[user@host ~]$ sudo deluser helpdesk itstaff
```

* Reads configuration in /etc/adduser.conf and /etc/deluser.conf

users

- Purpose
  - Print the usernames of users currently logged in
- Syntax

```
users [options] [file]
```

- Examples

```
# Print currently logged users
[user@host ~]$ users
```

w

- Purpose
  - Show who is logged on and what they are doing
- Syntax

```
w [options] user
```

- Examples

```
# Print information about the logged on users
[user@host ~]$ w
# Print shorter version
[user@host ~]$ w --short
```

who

- Purpose
  - Show who is logged on
- Syntax

```
who [options] [file | arg1 arg2]
```

- Examples

```
# Print currently logged users with headers
[user@host ~]$ who -Hu
```

whoami

- Purpose
  - Print effective userid
- Syntax

```
whoami [options]
```

- Examples

```
# Print the effective user
[user@host ~]$ whoami
```

last

- Purpose
  - Show listing of last logged in users
- Syntax

```
last [options]
```

- Examples

```
# List the last five lines
[user@host ~]$ last -n 5
# Print full login and logout times and dates
[user@host ~]$ last -F
```

lastb

- Purpose
  - Show listing of last unsuccessful login attempts
- Syntax

```
lastb [options]
```

- Examples

```
# List the last five lines
[user@host ~]$ sudo lastb -n 5
# Display full user and domain names
[user@host ~]$ sudo lastb -w
```

lastlog

- Purpose
  - Report most recent login for all users
- Syntax

```
lastlog [options]
```

- Examples

```
# List users and the last time they logged in
[user@host ~]$ lastlog
```

passwd

- Purpose
  - Update user's authentication tokens
- Syntax

```
passwd [options] [login]
```

- Examples

```
# Change password for the logged user
[user@host ~]$ passwd
# Change password for another user
[user@host ~]$ sudo passwd username
```

chpasswd*

- Purpose
  - Update passwords in batch mode
- Syntax

```
chpasswd [options]
```

- Examples

```
# Change password for a user
[user@host ~]$ echo username:password | sudo chpasswd
```

* Not installed by default on the recent Red Hat-based distributions. Available via the **shadow-utils** package

chage*

- Purpose
  - Change user password expiry information
- Syntax

```
chage [options] login
```

- Examples

```
# Show account aging information
[user@host ~]$ chage -l user
# Set expiry date for an account
[user@host ~]$ sudo chage -E 2019-12-31 username
```

* Not installed by default on the recent Red Hat-based distributions. Available via the **shadow-utils** package

chfn*

- Purpose
  - Change user finger (descriptive) information
- Syntax

```
chfn [options] [login]
```

- Examples

```
# Change finger information for the current user
[user@host ~]$ chfn
# Set full name and office of a user
[user@host ~]$ sudo chfn -f 'User 2' -o 'IT' user2
```

* Not installed by default on the recent Red Hat-based distributions. Available via the **util-linux-user** package

chsh*

- Purpose
  - Change user shell
- Syntax

```
chsh [options] [login]
```

- Examples

```
# List available shells
[user@host ~]$ chsh --list-shells
# Change the shell of a user
[user@host ~]$ sudo chsh -s /bin/sh user2
```

* Not installed by default on the recent Red Hat-based distributions. Available via the **util-linux-user** package

groupadd

- Purpose
  - Create a new group
- Syntax

```
groupadd [options] group
```

- Examples

```
# Add group and assign the next available id
[user@host ~]$ sudo groupadd accounting
# Add group with custom id
[user@host ~]$ sudo groupadd -g 2000 developers
```

groupmod

- Purpose
  - Modify a group definition on the system
- Syntax

```
groupmod [options] group
```

- Examples

```
# Rename a group
[user@host ~]$ sudo groupmod -n newname oldname
# Change group id
[user@host ~]$ sudo groupmod -g 1500 accounting
```

groupdel

- Purpose
  - Delete a group
- Syntax

```
groupdel [options] group
```

- Examples

```
# Delete a group
[user@host ~]$ sudo groupdel accounting
```

addgroup

- Purpose
  - Create a new user or system group*
- Syntax

```
addgroup [options] group
```

- Examples

```
# Create new user group
[user@host ~]$ sudo addgroup itstaff
# Create new system group
[user@host ~]$ sudo addgroup --system daemons
```

* Reads configuration in /etc/adduser.conf

delgroup

- Purpose
  - Remove user or system groups*
- Syntax

```
delgroup [options] group
```

- Examples

```
# Remove user group
[user@host ~]$ sudo delgroup itstaff
# Remove system group
[user@host ~]$ sudo delgroup --system daemons
```

* Reads configuration in /etc/adduser.conf and /etc/deluser.conf

groups

- Purpose
  - Print the groups a user is in
- Syntax

```
groups [options] [username]
```

- Examples

```
# Print list of groups to which a user belongs
[user@host ~]$ groups
```

gpasswd

- Purpose
  - Administer groups and their passwords
- Syntax

```
gpasswd [options] group
```

- Examples

```
# Change password of a group
[user@host ~]$ sudo gpasswd developers
# Set a user as administrator for a group
[user@host ~]$ sudo gpasswd -A user developers
```

newgrp

- Purpose
  - Log in to a new group
- Syntax

```
newgrp [-] [group]
```

- Examples

```
# Change current group
[user@host ~]$ newgrp developers
# Simulates user login while changing the group
[user@host ~]$ newgrp - developers
```

id

- Purpose
  - Print real and effective user and group IDs
- Syntax

```
id [option] [user]
```

- Examples

```
# Print user and group information for current user
[user@host ~]$ id
# Print group IDs of a user
[user@host ~]$ id -G newuser
```

sudo

- Purpose
  - Execute a command as another user
- Syntax

```
sudo [options]
```

- Examples

```
# Execute command as root
[user@host ~]$ sudo useradd testuser
# Execute command as another user
[user@host ~]$ sudo -u helpdesk ls /home/helpdesk
```

su

- Purpose
  - Run a command with substitute user and group ID
- Syntax

```
su [options] [-] [user]
```

- Examples

```
# Switch to a user
[user@host ~]$ su helpdesk
# Switch to a user with login shell
[user@host ~]$ su - helpdesk
```

Access Rights

Mechanics and Management

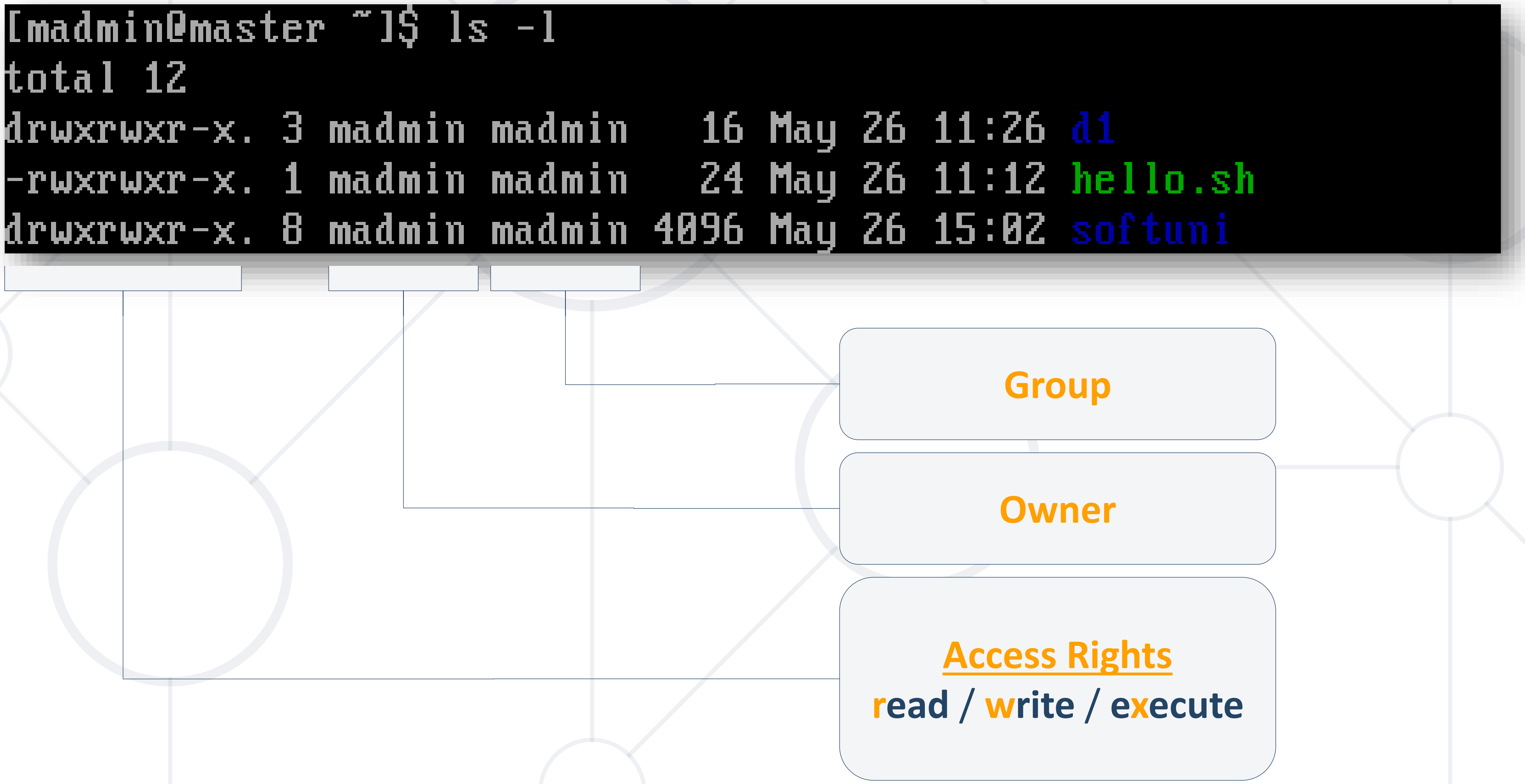
A Few Words On Security

- Two levels
  - **Level 1**: **Discretionary Access Control** (**DAC**)
    - Regular **file access permissions** *
    - Access Control Lists (**ACL**) **
  - **Level 2**: **Mandatory Access Control** (**MAC**)
    - Typical examples - **SELinux** and **AppArmor**
- Applied from **Level 1** to **Level 2**

* Current process UID and GID are compared with the UID and GID of the file being accessed with regards to the permissions set

** ACL is a list of permissions attached to an object in the file system. It extends standard permissions and allows more options

Access Rights



```
[madmin@master ~]$ ls -l
total 12
drwxrwxr-x. 3 madmin madmin   16 May 26 11:26 d1
-rwxrwxr-x. 1 madmin madmin   24 May 26 11:12 hello.sh
drwxrwxr-x. 8 madmin madmin 4096 May 26 15:02 softuni
```

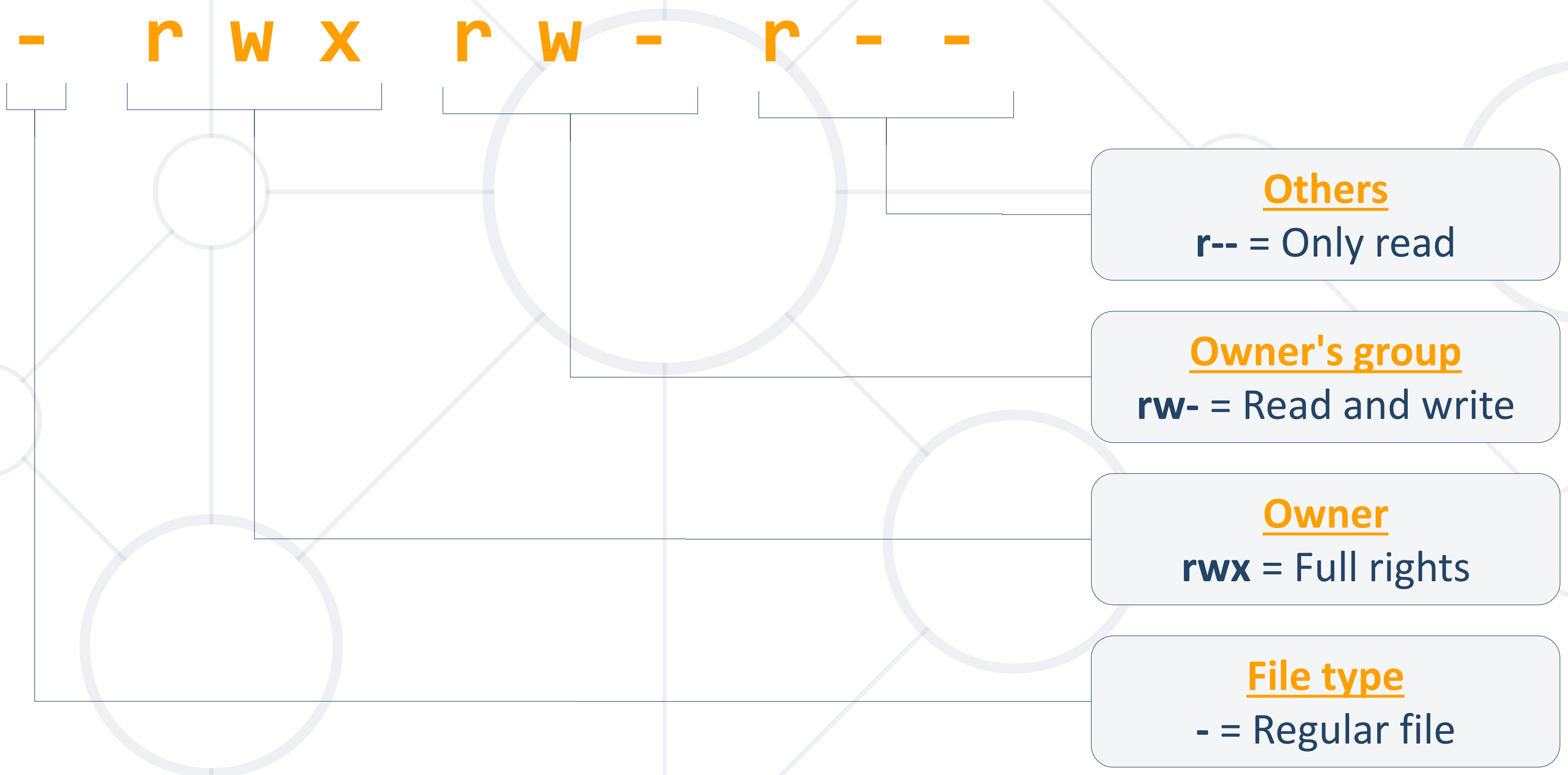
Access Rights – Meaning

- **Read**
  - **Files** - allows a user to view the contents of the file
  - **Directories** - allows a user to view the names of the file in the directory
- **Write**
  - **Files** - allows a user to modify and delete the file
  - **Directories** - allows a user to delete the directory, modify its contents (create, delete, and rename files in it), and modify the contents of files that the user can read

Access Rights – Meaning

- **Execute**
  - **Files** - allows a user to execute a file (the user must also have read permission)
  - **Directories** - allows a user to access, or traverse, into and access metadata about
    files in the directory

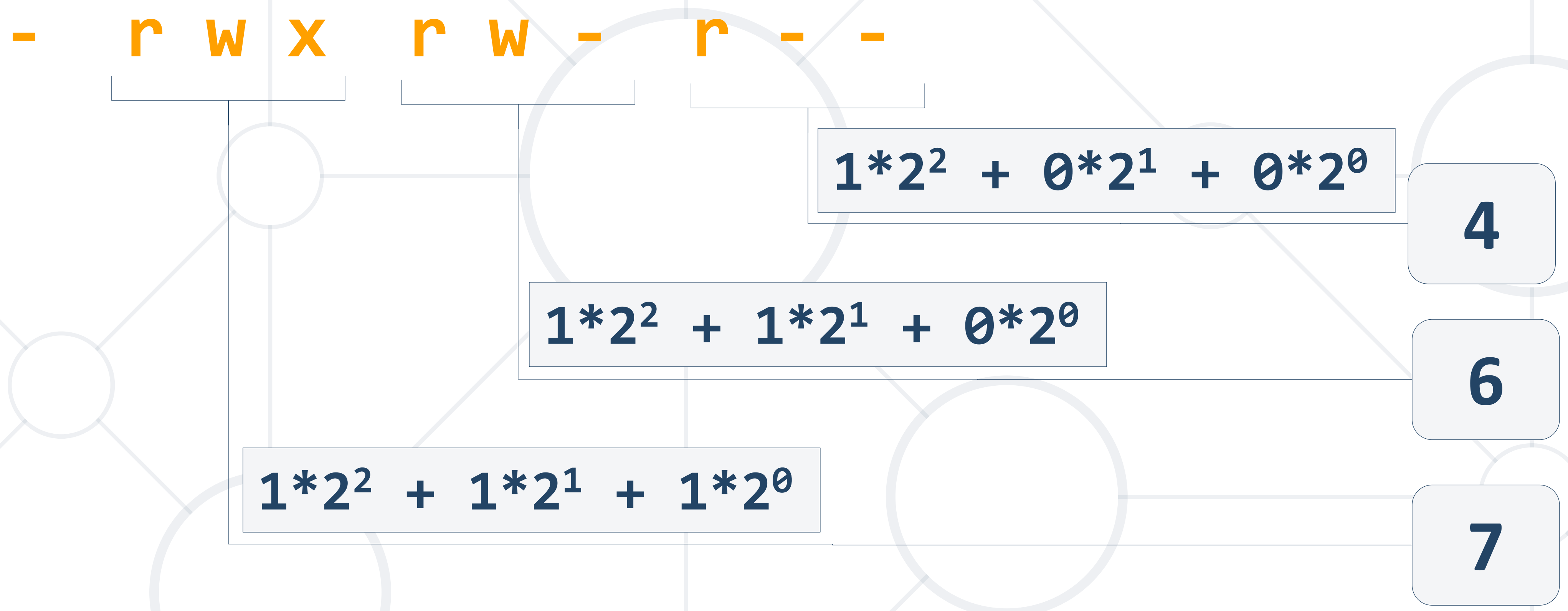
Access Rights – Symbolic Notation (*,**)



* When using symbolic notation individual permissions can be granted or revoked

** Permissions changes get active immediately

Access Rights – Octal Notation (*, **)



Weights: read - 2, write - 1, execute - 0

* When using octal notation permissions are set as a whole, not individually

** Permissions changes get active immediately

Access Rights – Notations Side by Side

Permissions	Binary	Octal	Description
---	000	0	No permissions
--x	001	1	Execute-only permission
-w-	010	2	Write-only permission
-wx	011	3	Write and execute permissions
r--	100	4	Read-only permission
r-x	101	5	Read and execute permissions
rw-	110	6	Read and write permissions
rwx	111	7	Read, write, and execute permissions

Default Access Rights

- For **files**
  - **Maximum** rights = **666**
  - Subtract **umask** = **002**
  - Result = **664**
- For **directories**
  - **Maximum** rights = **777**
  - Subtract **umask** = **002**
  - Result = **775**

```
   6   6   6
-
   0   0   2
  ___________
   6   6   4
(rw- rw- r--)
```

```
   7   7   7
-
   0   0   2
  ___________
   7   7   5
(rwx rwx r-x)
```

Special Permissions – Sticky Bit

- Prevent non-owners of a file to delete it
- Usually used for directories
- Numeric permission is **1xxx**
- Can be set in both ways

```
# Set sticky bit of a folder with permissions 755
[root@host ~]# chmod 1755 /dir

# Set sticky bit using a symbolic notation
[root@host ~]# chmod o+t /dir
```

Special Permissions – Set Group ID (SGID)

- Allows users to run a program as if it was member of the group
- Usually used for directories
- All new files are owned by the group
- Numeric permission is **2xxx**
- Can be set in both ways

```
# Set SGID to a file with permissions 644
[root@host ~]# chmod 2644 script.sh
# Set SGID using a symbolic notation
[root@host ~]# chmod g+s script.sh
```

Special Permissions – Set User ID (SUID)

- Allows users to run a program as if it was its owner
- Usually, the owner is root
- Numeric permission is **4xxx**
- Can be set in both ways

```
# Set SUID to a file with permissions 644
[root@host ~]# chmod 4644 script.sh

# Set SUID using a symbolic notation
[root@host ~]# chmod u+s script.sh
```

chmod

- Purpose
  - Change file mode bits
- Syntax

```
chmod [options] file
```

- Examples

```
# Set fixed permissions*
[user@host ~]$ chmod 755 hello.sh
# Remove execute permission for the group*
[user@host ~]$ chmod g-x hello.sh
```

* No **sudo** is needed when objects are owned by the current user

chown

- Purpose
  - Change file owner and group
- Syntax

Can be replaced with "."

```
chown [options] [owner][:[group]] file
```

- Examples

```
# Change both owner and group of a file*
[user@host ~]$ chown user:users file.txt
# Change recursively the group for a folder*
[user@host ~]$ chown -R :developers project/
```

* No **sudo** is needed when objects are owned by the current user

chgrp

- Purpose
  - Change group ownership
- Syntax

```
chgrp [options] group file
```

- Examples

```
# Change the group of a files*
[user@host ~]$ chgrp developers code*
# Change recursively the group for a folder*
[user@host ~]$ chgrp -vR developers project/
```

* No **sudo** is needed when objects are owned by the current user

umask

- Purpose
  - Display or set file mode mask
- Syntax

```
umask [options] [mode]
```

- Examples

```
# Show current mask using the symbolic notation
[user@host ~]$ umask -S
# Set new mask
[user@host ~]$ umask 0022
```

Problem: Add User

- Conditions
  - Existing folder (**/users/newuser**) owned by **root**
  - Existing group (**projectx**)
- Goals
  - Username: **newuser**
  - Password: **Password1**
  - Home directory: **/users/newuser**
  - Explicit shell: **/bin/bash**
  - Member of the **projectx** group

Solution: Add User

```
[root@host ~]# useradd -d /users/newuser -s /bin/bash
-G projectx newuser
...
[root@host ~]# passwd newuser
...
```

- If the specified directory **exists**, then:
  - Files from /etc/skel **won't be copied** as usual
  - We should take care of the **user access rights**
  - We should organize **environment initialization files** for the user

Practice

Summary

- Environment defines the operational conditions
- Environment is globally set-up, modifiable, inheritable
- Executables are Built-in, External, Aliases, and Functions
- Execution order can be followed or ignored
- Shell is configured by multiple files. The configuration depends on the shell
- Many sources of help are available off-line on the system
- (Almost) everything on Linux system is files
- Users and groups are used to control access to the system in general
- Access rights and ownership are assigned to users and groups

Resources

- openSUSE Help
  - https://doc.opensuse.org/
- Debian Help
  - https://www.debian.org/doc/
- Red Hat (AlmaLinux/Rocky Linux) Help
  - https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/

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