Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

Legal note

These slides form an outline of the curriculum and contain multiple references to the official Red Hat training materials (codes RH124, RH134 and RH254), since students of this subject are also eligible for the named RH learning materials. Some diagrams and pictures originating from the RH courses will show up in this outline as there is an agreement in place which allows for our Faculty to teach this subject based on the Red Hat curriculum.

All other pictures and diagrams sourced from third parties are explicitly marked with a reference to the origin.

Be aware that the mentioned parts and also this series of slides as a whole are property of their respective owners and are subject to copyright law.

Legal note

All students of PTE-MIK who have officially taken the course „Linux System Administration” are eligible to download these slides from the Faculty’s internal network, and are eligible for license keys for Red Hat System Administration online learning materials at rhlearn.gilmore.ca as part of their training program.

Do not share, redistribute, copy or otherwise offer these learning support materials, license keys or account information either for money or free of charge to anyone, as these materials contain **intellectual property.**

Unauthorized distribution is both against the law and university policy and will result in an in-campus inquiry, suing for damages and/or criminal prosecution by the University of Pécs, Red Hat Inc. and other parties.

Local users and groups

Lecture 3

Chapter 1

Users

- All user activity, all the terminals, all the processes and all the files in the system are tied to a specific user.
- Users are accounted and internally tracked by a unique numeric identifier, called **UID**.
- Software output often shows the users name instead of the ID. This is only for convenience, internal structures store the numeric ID. The mapping of UID to username is world readable public information.
- The public part of a user account data is stored in the **/etc/passwd** file.
- The confidential part of user information is stored in **/etc/shadow** (for e.g. password hashes are not public)
- You can get information about the currently logged in user with the **id** and **whoami** commands.

Users and Groups

- You can ask for the list of other users currently logged in to the system with the **w** or **who** commands.
- You can check the last login time of each user with **lastlog**, and you can print the most recent logins to system the with **last** cammand.
- You can list the running processes with ps. Using the **-u** switch turns on the user column.
- Groups are container objects of users for organizational purposes. All users have exactly one primary group, but additionally can be members of any number of groups.
- The default is to have a UPG (User Private Group), which means the group has only one user as a member and the group has the same name as the user.
- New files created by users will belong to their respective primary groups.

Groups

- Every filesystem object has exactly one user and one group defined as owner.
- A user can have any number of supplementary groups and groups can have any number of members.
- Groups are also tracked and identified by a numeric value called **GID**.
- Group membership is public information, it is stored in the **/etc/group** file.
- Groups can also have passwords. This is confidential, therefore separated to the **/etc/gshadow** file.

- Refer to **RH124 CH5.1.** for further information.

UID conventions

- UID of 0 (zero) is always assigned to the administrative superuser called 'root'.
- UIDs 1-999 are assigned to system users. These are unprivileged users, usually assigned to daemon processes in order to limit their access to a specific subset of tasks/files. They usually can’t even log in to a shell session in a regular way.
- A subset of system users (1-200) are statically assigned by RedHat, the rest can be assigned dynamically.
- UIDs starting from 1000 are available for regular (human) users.
- A special UID of 65534 is assigned to the most unprivileged user, called 'nobody'. This is often needed to map privileges of network users to remote filesystems.

Local user accounts

- The source of user and group accounting information is a database. It can be stored and distributed through several protocols and mechanism, even over a network.
- In this chapter only the locally defined accounts will be discussed.
- Data is stored in tables, that can be viewed or edited with an editor. These are simple text files (file names shown earlier in this chapter) that contain data related to one user per line. Fields are separated with a : (colon).

/etc/passwd

```
jsmith:x:1003:1003:John Smith:/home/jsmith:/bin/zsh
jdoe:x:1004:1004:Jane Doe:/home/jdoe:/bin/bash
```

/etc/group

```
students:x:1010:bob,alice,jsmith,jdoe
```

The passwd file

- User name
- Password field, x means an encrypted password has to be looked up elsewhere → /etc/shadow
- UID (numeric)
- GID of primary group (numeric)
- GECOS (freely usable text field: comments, location, real name, etc.)
- Path to the user's home directory
- Shell (It doesn't have to be an actual shell, note for e.g. /bin/nologin or /bin/false)

/etc/passwd

```
jsmith:x:1003:1003:John Smith:/home/jsmith:/bin/zsh
jdoe:x:1004:1004:Jane Doe:/home/jdoe:/bin/bash
```

The group file

- Group name
- Group password field, x means to look for encrypted password hashes elsewhere → /etc/gshadow
- GID (numeric)
- Enumeration of all the members. Member user names are separated by a , (colon).

/etc/group

```
students:x:1010:bob,alice,jsmith,jdoe
```

The shadow file

- User name (as string, not numeric ID)
- Encrypted password: has several sub-fields (next slide)
- Last change date (days since Epoch)
- Minimum days before user can change password
- Maximum days before user must change password
- Nr. of days to warn user before obligatory pw. change
- Account disable timeout (in days from pw expiry)
- Days since account has been disabled
- Reserved field

/etc/shadow

```
jsmith:$1$gCjLa2/Z$6Pu0EK0AzfCjxjv2hoLOB/:10063:0:
99999:7:::
```

The shadow file’s password subfields

- The password field has several subfields, separated by $ (dollar) sign: $hash_algo$salt(optional)$pwhash
- Hash algo examples: 1 for MD5, 5 for SHA256, 6 for SHA512
- Salt is a random sequence, hash is function of both salt and password string. → Note users with same password
- An ! (exclamation mark) in the password field indicates the accounts password is locked. (SSH keypair login is still possible, password login is not possible)
- An * (asterisk) in the password field indicates the account has been locked. (No login possible)
- An empty password field indicates the user can log in without entering a password. (Guest users)

The gshadow file

- Group name (as string, non numeric)
- Encrypted password
  - ! means password group login (newgrp) is disabled
  - If set to a hash, users can log in the group with a valid password.
- List of group administrators (comma separated)
  A group admin can change the group password and can add/remove members . (gpasswd)
- List of members (comma separated)
  Members can log into the group without typing a password even if password is set. (newgrp)

```
/etc/gshadow
students:!!:instructor:bob,alice
```

Managing users

- You can manage before-mentioned text files manually. Or you can use the following command line tools. This conveniently also creates a home directory and sets its permissions, besides adding a new user entry.
- The **useradd [switches] username** command can create a user, note the following switches and default values
  - -c to set comment (Gecos) field (empty)
  - -s to specify shell (/bin/sh)
  - -d to specify home directory (/home/[username])
  - -g to specify primary group (UPG)
  - -G to specify list of supplementary groups (none)
  - -u to specify UID (add one to the last used UID)

Shell command line

```
# useradd -c "John Smith" -u 1050 -G video jsmith
```

Managing users

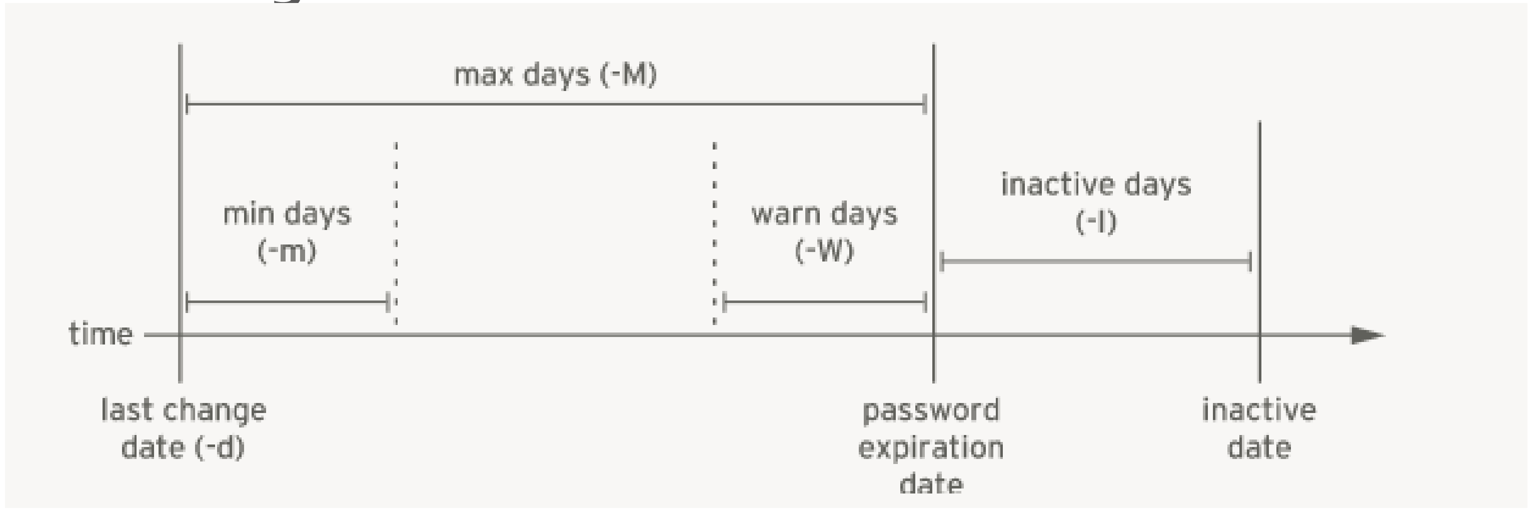
- You can modify an existing user with **usermod [switches] username** using similar switches. Also note the -L and -U switches to lock and unlock the password.
- The **userdel username** command removes the user. Using the -r switch also removes the users home directory from the filesystem.
- Note the seemingly conflicted UIDs in **RH124 CH 5.4.** when deleting and adding users.
- You can change your own password with **passwd** command. Only root can change passwords for other users. Root can do it like this: **passwd alice**
- Complete lab practice under **RH124 CH 5.5.**

Managing groups

- You can add a new group with **groupadd** command.
- You can modify a group with **groupmod**.
- You can remove a group with **groupdel**.
- You can modify group membership with the usermod command as described earlier. Note the -g -G and -a command switches.
- Refer to **RH124 CH 5.6.** for further
- Complete lab practice under **RH124 CH 5.7.**

Managing password aging

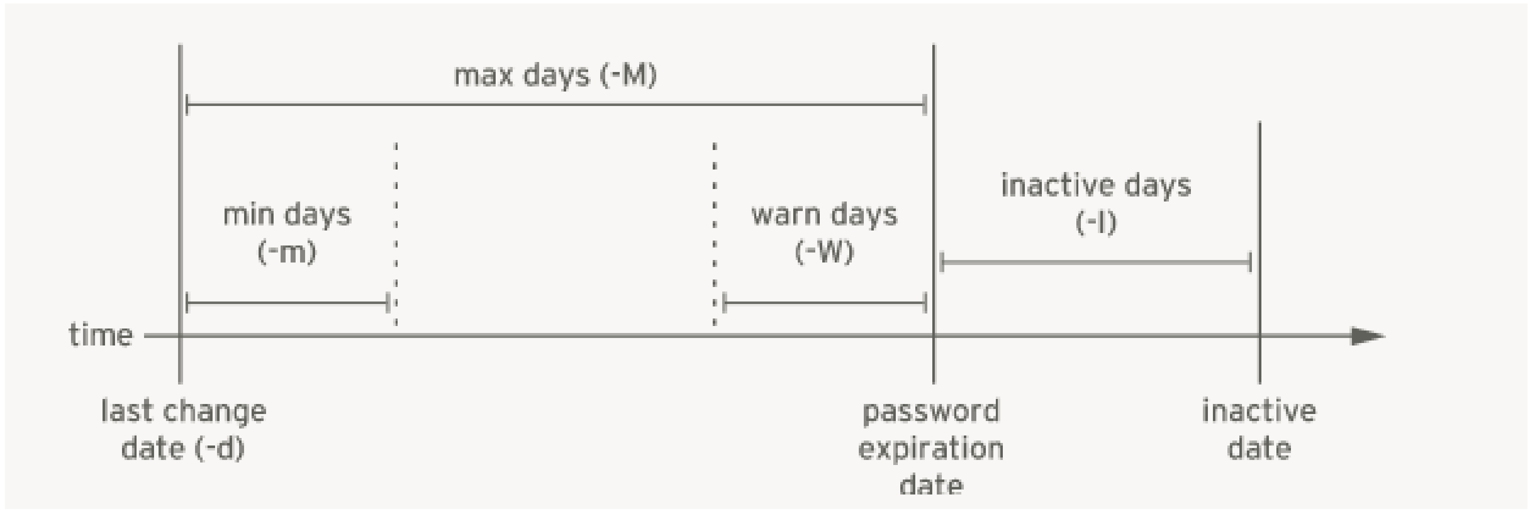
- You can adjust the timing fields of the /etc/shadow file with **chage**.



- Use chage -E to set absolute account expiry.
- You can also use the usermod -e command to set expiry.
- Also note the -I, -d options to chage.
- Refer to **RH124 CH5.8.-5.9.** for further information.
- Complete practice **RH124 CH5.10.**

Managing password aging

- You can adjust the timing fields of the /etc/shadow file with **chage**.



- Use chage -E to set absolute account expiry.
- You can also use the usermod -e command to set expiry.
- Also note the -l, -d options to chage.
- Refer to **RH124 CH5.8.-5.9.**for further information.
- Complete practice **RH124 CH5.10.**

File access permissions

Lecture 3

Chapter 2

Permission and owner information

- As stated earlier in this lecture files have exactly one owner and one group owner. Processes also have owners and group owners.
- A file can only be accessed by a process if it is permitted by comparing their ownership information and evaluating file permissions.
- In *nix systems permissions are organized in 3 sets for each filesystem object. One set is for the user owner (u), one set for the group owner (g) and one set for all the other users (o).
- Each set contains three binary values, each determining one of whether or not **r**ead, **w**rite and e**x**ecute operation is permitted on the file.
- There are some more special bits for each file that are not part of the three sets. (sticky, setgid, setuid)

Permissions explained

- Permission sets can be printed by adding the -l switch to the **ls** command.

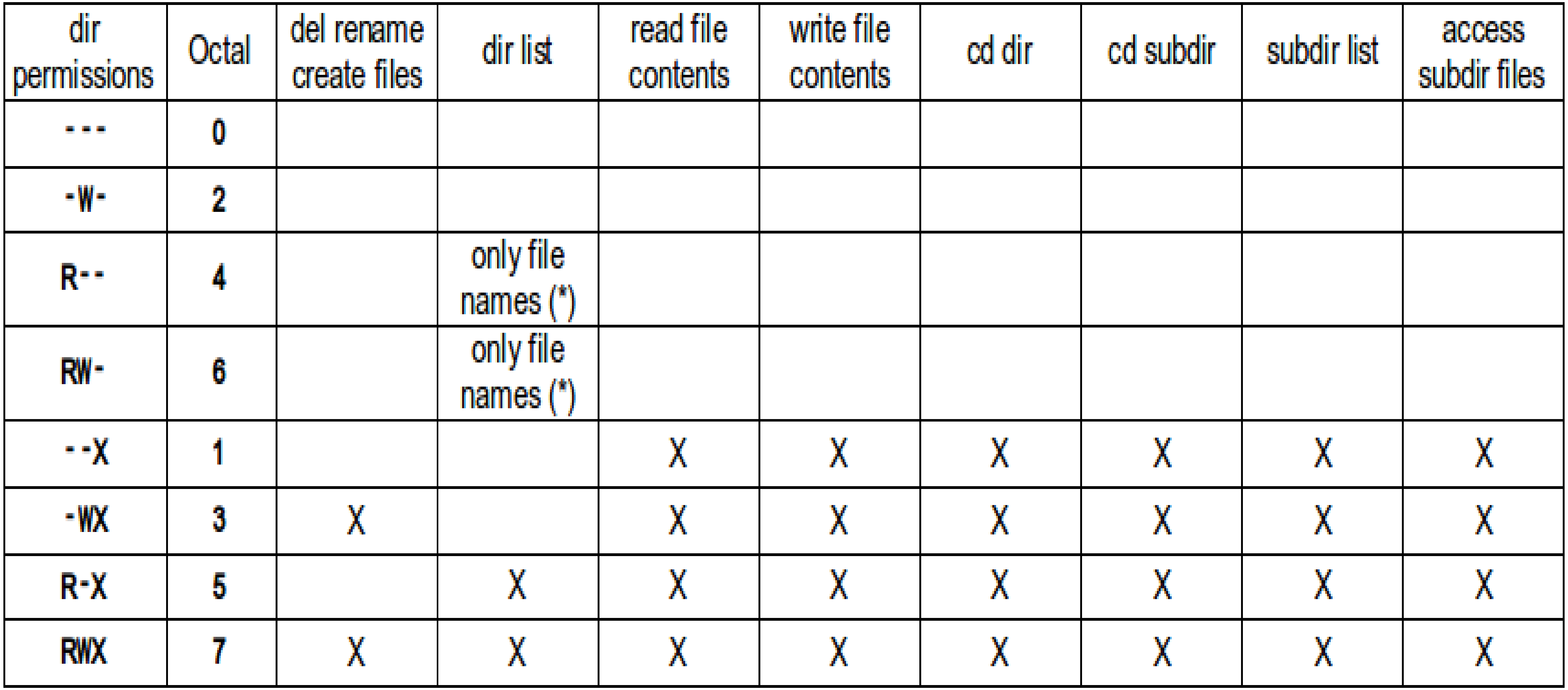
Shell command line

```
# ls -l /var/lib
drwxr-xr-x.  2 root    root   4096 2014 ápr   14 bluetooth
drwxr-xr-x.  2 chrony  chrony4096 2014 szept 10 chrony
drwxr-xr-x.  3 root    root   4096 2013 dec   12 color
drwxr-xr-x.  3 colord  colord4096 2014 nov    5 colord
drwx------.  2 apache  apache4096 2014 dec   17 dav
```

- Read and write permissions are self-explanatory for files. Execute permission allows for text scripts or ELF binary files to be started (a process created from them)
- Keep in mind, that directories are simple lists only. They list filename-to-inode mappings. (Further in Lecture 5)
- Read permission of directories allows for reading the list itself. (listing)

Permissions explained

- Write permission of a directory allows for renaming, adding or deleting files within the directory. (Editing the list). Write only works with execute bit also set.
- Execute permission of a directory allows for entering it with **cd**, and accessing its contained objects.
- Having execute without read for directory makes it impossible to list the objects inside. On the other hand you can still read/access such objects, if you know them to be there and know their name exactly.
- Having read without execute makes listing possible, but objects can not be accessed inside such directories. E.g. ls works, but with no details are shown (no access to inodes). Also no files can be read or modified.
- Refer to **RH124 CH6.1-CH6.3.** for further details.

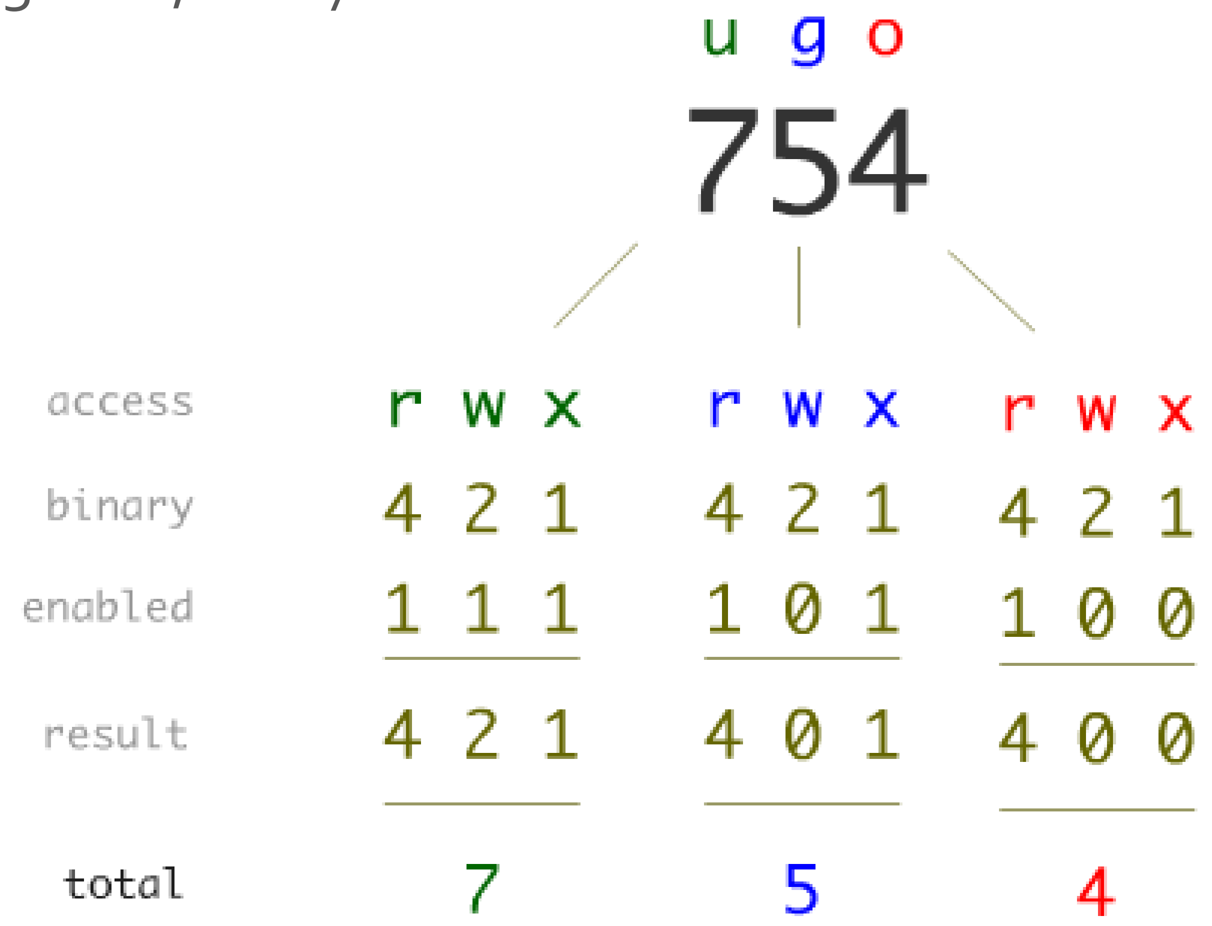


dir permissions	Octal	del rename create files	dir list	read file contents	write file contents	cd dir	cd subdir	subdir list	access subdir files
---	0								
-W-	2								
R--	4		only file names (*)						
RW-	6		only file names (*)						
--X	1			X	X	X	X	X	X
-WX	3	X		X	X	X	X	X	X
R-X	5		X	X	X	X	X	X	X
RWX	7	X	X	X	X	X	X	X	X

https://i.stack.imgur.com/mZ6qv.png

Octal (numerical) representation

- The octal representation of permissions is also commonly used. Values for special bits are: setuid – 4, setgid – 2, sticky – 1



https://devopscube.com/wp-content/uploads/2016/05/permissions.png

Special permissions

- The **sticky bit** can only be effective with the executable bit set. For files it means to keep the binary in swap for performance tuning. For directories that are world writable (others have w bit) it restricts removing/ renaming objects to the owners of said object or the owner of the sticky directory itself. (E.g. /tmp)
- The **setuid/setgid** bits only make sense in conjunction with the executable bit set. A process created from such a file will not have the owner of the user session which executed it, rather it will have the owner and group of the filesystem object instead. Applying the setgid bit to directories forces the group of directory and the setgid bit itself to be inherited to newly created objects inside the directory. Setuid can work similar for the owner but on some systems it may have no effect.
- Note the **ls -l** representation (s,S,t,T)

Modifying owner and permissions

- You can use the **chmod** command to change permissions of a file. Note the different representations accepted by chmod. E.g. chmod 0755 file, chmod o=rx file, chmod a+w file, chmod -222 file
- The owner can be changed with **chown.** Consider the following forms: chown student file, chown student:class file, chown :class file
- Only the owner can change the file mode. Only root can change the owner of a file (no give-away possible). Group owner can be set by users who are members of both the original and newly set group.
- You can set the default permissions for new objects with the **umask** command. Specify a mask that is subtracted from 777 → the result will be applied to all newly created objects. (also see bashrc)
- Complete practice **RH124 CH6.4.** through **CH6.6.**

Teamwork example

- If you had two groups of people, let's say mechanical engineers and electrical engineers and wanted them to have separate directories which are shared among members but not across groups, you could follow this:
- Add each engineer to either the electrical or mechanical group. Create one directory owned by the group for each of the groups. Set the setgid bit for the directories. Set mode for all files and sub-directories to 770. This way all **new** files will be read/writable to the respective group. A specific user can be member of both groups, but created files will only be accessible for one of the groups, the users primary group.
- Now if you wanted to share something between the two groups, you would have to create an additional directory with an additional group, which has all the engineers as members. Than have all users use newgrp accordingly.

ACLs

- As the number of combinations of possible group memberships grow, this can become quite complex.
- Also it wouldn’t be easy to specify negative rights with the the standard permission bits. For e.g.: every electrical engineer is allowed except for bob.
- That’s where ACLs are more relevant. Access control lists are policies made up of several rules. Each rule can target a specific user or a specific group.
- Each file can have an ACL consisting of any number of rules.
- If ACL is present on a file it will have 3 mandatory elements which can not be deleted, only modified. One for the owner, for the group owner and one for others. These three are called base ACLs, they are copied from standard permissions upon ACL initialization.

ACL mask

- The base ACLs for owner and others are 1:1 equivalents to standard permissions, chmod actually changes these two base ACLs as well, and changing these ACL entries is represented by file mode. (works vice-versa)
- What ACLs do, is that they override the standard 3-bit group permission mechanism with a set of named rules targeting users and/or groups, including the group owner itself.
- The ACL **mask** only affects the mentioned part of entries, specifically named user and named group entries plus the group owners entry. It does not affect the owning user part and the 'others' part of ACLs.
- The mask is applied as a bit-wise **AND** operand to the subset of rules mentioned before, resulting in the effective ACLs.

ACL entries

- A + sign in ls output shows that ACLs are set for a file. In this case the 3-bit group permissions would make no sense, since there is a list of rules instead → the **middle triplet** shows the ACL mask, and chmod-ing the middle triplet changes the mask **instead of group** permissions.

Shell command line

```
# ls -l /home/student/Documents
drwxr-x---+  2 student school 4096 2017 dec 12 thesis.txt
```

- A file has exactly three base ACLs. Each file can have one or zero mask entry. A mask entry is mandatory if any non-base ACLs are set.
- Besides that a file can have any number of named entries. A named entry can specify a name string or a numeric identifier as well.
- A directory can also have default ACLs. These are automatically inherited to newly created sub-items.

ACL evaluation

- The evaluation order is the following:
  - If the accessing user owns the file → the user ACL (triplet) applies. (just like with standard permissions)
  - If the the accessing user has a named entry → it will be applied.
  - If the accessing users group own the file → the group ACL is applied.
  - If the accessing group has a nemd entry → it will be applied.
  - If not any of the above rules match → the 'other' ACL will be applied. (just like with standard permissions)
- For further on ACLs, read **RH134 CH6.1.**
- For setting and checking ACLs refer to **RH134 CH6.2.** and the following three slides.

Reading ACLs

Shell command line

```
# getfacl some_directory
# file: some_directory
# owner: student
# group: controller
# flags: -s-
user::rwx
user:james:---
user:1005:rwx           #effective:rw-
group::rwx              #effective:rw-
group:sodor:r--
group:2210:rwx          #effective:rw-
mask::rw-
other::---
default:user::rwx
default:user:james:---
default:group::rwx
default:group:sodor:r-x
default:mask::rwx
default:other::---
```

Setting ACLs

- An ACL can be set/replaced with the **setfacl -s** command. Use the -m switch to modify an ACL entry, use -x to remove an entry. Use the -d switch together with the former two to modify/remove a default entry.
- The format starts with u: g: o: or m: for user, group, other, and mask entries respectively.
- The next element is an object specifier (user/group name/id). **Mask** and **other** entries have blank specifiers.
- An emtpy user or group specifier corresponds to the user/group who owns the file.
- The last element is the permission set itself. e.g rx

Shell command line

```
# setfacl -m u::rwx,g:sodor:rX,o::- filename
# setfacl -x -d u:james filename
```

Setting ACLs

- You can use the -R switch to apply ACLs to directories and sub-entries recursively. A permission set containing a X means to apply x permission only to directories, not files.
- You can remove all default ACL entries of a directory with setfacl using the -k switch. The -b switch removes ACLs completely.
- Modifying the ACLs with **setfacl** can cause the mask to be recalculated. To inhibit mask change use the -n switch to setfacl.
- ACLs somewhat overlap with Linux permissions, it is recommended that you don't use chmod anymore on files which have ACLs. It is recommended to use **setfacl** instead.
- Complete practices **RH134 CH6.3.** and **6.4.**

PAM, nsswitch

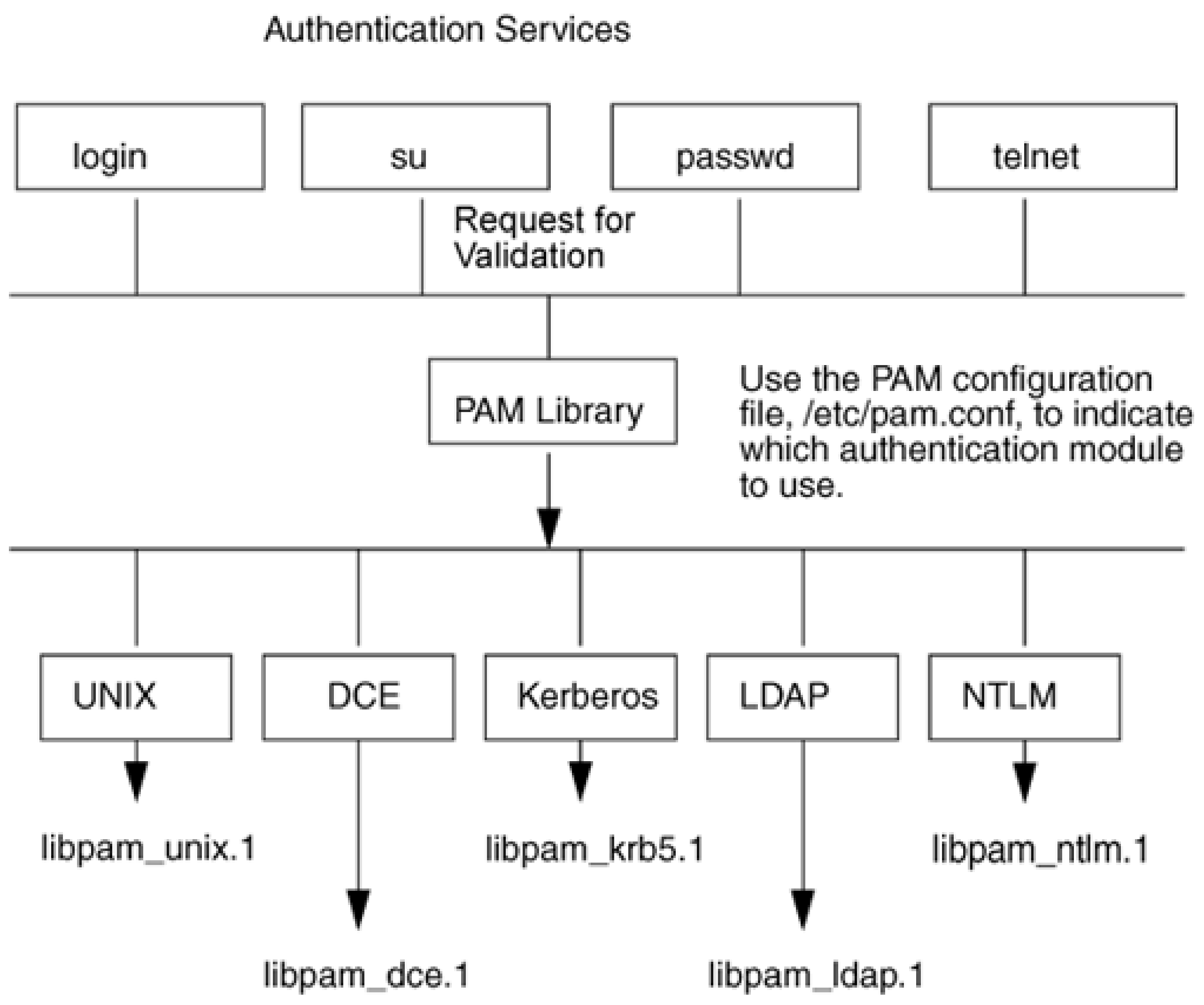
Lecture 3

Chapter 3

PAM

- Pluggable Authentication Modules apply an abstraction layer to authentication mechanism of Unix.
- Applications send a request for authentication to PAM.
- PAM then performs the authentication with several steps and checks, in a configurable way. This involves using modules.
- When all the necessary modules are done, a response is returned to the application.
- In this way any changes to the authentication scheme induces a reconfiguration of PAM but not the reconfiguration/recompilation of applications themselves.

PAM diagram



https://docstore.mik.ua/manuals/hp-ux/en/5992-3387/img/gfx1.png

PAM module examples

- pam_time: decides whether a users request for authentication is allowed at a given time from a given terminal. (/etc/security/time.conf)
- pam_securetty: only allows root to log in from a subset of terminals, which are considered secure. The set is listed in /etc/securetty file.
- pam_unix: checks if entered password corresponds to hash stored in /etc/shadow.
- pam_nologin: denies login if /etc/nologin file exists.
- pam_allow: unconditional allow, always returns true.
- pam_deny: unconditional deny, always returns false.
- pam_warn: write authentication request and details in system logs.
- pam_pwdb, pam_limits, pam_env, etc...

PAM modules

- There are four activity types of PAM modules:
  - **authentication** modules verify user identity, typically by asking for a key, password, secret, fingerprint, biometric characteristic, something only a valid user would know/posses.
  - **session** modules perform actions after authentication (session start) and on logout/loss of connection (session end). E.g. print motd, mount home directory, etc…
  - **account** modules check if authentication target (user) is valid under the given conditions. (e.g. time of day, terminal, IP address, etc.)
  - **password** modules take part in updating authentication credentials (e.g. enforcing strong password on pw change.)

PAM control flag

- There are four control flags possible for modules:
  - **requisite**: If such a module fails, PAM will immediately return failure without any further module calls.
  - **required**: If such a module fails, PAM will return failure, but will continue to call remaining modules in the stack. (For e.g. log the issue, print something, etc.)
  - **sufficient**: If such a module succeeds, PAM will return pass, and stop calling any further module functions.
  - **optional**: such module's return value is ignored. Usually it stands for modules that should perform an action either way. (E.g. log the authentication attempt, regardless whether it fails or passes.)

PAM examples

- Prevent non-root users from shutting down the system:

```
/etc/pam.d/halt
auth          sufficient    pam_rootok.so
auth          required      pam_deny.so
```

- Enforce strong passwords

```
/etc/pam.d/system-auth
[...]
password     requisite     pam_passwdqc.so  min=12,10
,10,8,6 retry=3
[...]
```

- Allow only members of 'wheel' group to use su

```
/etc/pam.d/su
auth     sufficient     pam_rootok.so
auth     required       pam_wheel use_uid
auth       include        system-auth
```

PAM examples

- rlogin PAM example

/etc/pam.d/rlogin

```
auth     requisite /lib/security/pam_securetty.so
auth     requisite /lib/security/pam_nologin.so
auth     sufficient /lib/security/pam_rhosts\
   _auth.so
auth     required /lib/security/pam_unix.so
account  required /lib/security/pam_unix.so
account  required /lib/security/pam_time.so
password required /lib/security/pam_cracklib.so \
   retyr=1 type=UNIX minlen=10 ocredit=2 dcredit=2
password required /lib/security/pam_unix.so \
   use_authok shadow md5
session required /lib/security/pam_unix.so
session optional /lib/security/pam_motd.so \
   motd=/etc/motd
```

NIS, nsswitch

- On large networks, in case of large number of nodes, it could be troublesome for user/password/hostname/etc.. databases to be maintained, kept synchronized on all hosts.
- System settings, like user accounts can come from different sources. E.g. the local /etc/passwd file or DNS (Domain Name Service) or NIS (Network Information Serices) or directory services like freeIPA, LDAP, AD, etc…
- A network source might be more convenient in case of large node count.
- Look at /etc/nsswitch.conf for examples

NIS, nsswitch

- nsswitch.conf

/etc/nsswitch.conf

```
passwd:       compat
group:        compat
shadow:       compat

hosts:        files dns
networks:     files

protocols:    db files
services:     db files
ethers:       db files
rpc:          db files

netgroup:     nis
```

Recommended reading

Lecture 3

Chapter 4

Relevant chapters in RH:

- RH 124 CH5 (local users and groups)
- RH 124 CH6 (file access pemissions)
- RH 134 CH6 (ACLs)

Read by next lecture:

- RH 134 CH7 (SELinux)
- RH 124 CH7 (manage linux processes)
- RH 134 CH5 (nice)

Use your rhlearn.gilmore.ca login to access the learning materials.

Thank you for your attention!

Lecture 3, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK