laboratorio di sistemi operativi

introduzione a UNIX

Daniele Radicioni

$ whoami

- Daniele Radicioni
- ufficio numero 19 (primo piano, c.d. 'area nuova')
  http://www.di.unito.it/~radicion/
- e-mail: radicion@di.unito.it
- ricevimento: tendenzialmente al lunedì pomeriggio 14-16, ma su appuntamento
- NO consulenza via mail!!!
- materiale del corso disponibile su I-learn, sul sito "Laboratorio di Sistemi Operativi - Corso A Turno T1", abbreviato in SOA_lab_T1_21-22

di.unito.it
Dipartimento di Informatica

informazioni preliminari

regola di partizionamento

- Permane invariata la divisione fra corso A e corso B per le lezioni di teoria:
  - **Corso A: dalla A alla K**
  - **Corso B: dalla L alla Z**
- I laboratori del secondo anno sono quadruplicati, usando la seguente regola:
  - Turno T1 e T3: matricola dispari
  - Turno T2 e T4: matricola pari

Quindi ***appartiene al turno T1 chi ha cognome che inizia con lettera compresa fra A e K e con matricola dispari…***

SO A LAB - Turno 1: orari

- LABORATORIO: Linguaggio C e laboratorio Unix:
  - Turno T1:
    - Martedì: ore 9-12
    - Mercoledì: ore 14-17

Sistemi Operativi (corso A)

- Il corso è diviso in una parte di teoria e in una parte di laboratorio.
  - La parte di teoria introduce l'architettura e il funzionamento dei moderni sistemi operativi.
  - Nella parte di Laboratorio verranno insegnati i fondamenti del linguaggio C e i concetti di base sull'uso e sullo sviluppo di programmi nell'ambiente del sistema operativo Unix.

Sistemi Operativi (corso A)

- Nella prima parte del semestre le lezioni settimanali saranno suddivise fra teoria e linguaggio C.
  - Dopo le prime 5 settimane di C (30 ore), le ore settimanali di laboratorio saranno dedicate al sistema UNIX (altre 30 ore)
- Teoria: 60 ore in aula
- Linguaggio C: 30 ore in laboratorio
- UNIX: 30 ore in laboratorio

Modalità di Esame

- Discussione sull'esercitazione finale di laboratorio: obbligatoria. Si consegue un voto in trentesimi che farà media (pesata) con lo scritto.
- Scritto: obbligatorio. Per accedere allo scritto è necessario avere superato il laboratorio.
- Orale vero e proprio: facoltativo. Integra il voto ottenuto sostenendo le parti precedenti e permette di prendere un voto massimo di 30/30 e lode. Per sostenere l'esame orale è necessario aver preso almeno 26/30 allo scritto.

Modalità di Esame

- Sostenendo solo la discussione del laboratorio e lo scritto è possibile ottenere un punteggio fino a 30/30.
- Sostenendo la prova orale facoltativa, è possibile prendere anche 30 e lode pur non avendo conseguito il punteggio massimo nella prova scritta (ovviamente si può però anche essere bocciati…)

Modalità di Esame

- È possibile presentarsi a tutti gli appelli (cinque all'anno) e consegnare fino a tre scritti per anno accademico. La consegna di uno scritto annulla automaticamente qualsiasi voto conseguito in scritti precedenti.
  - Uno scritto nel quale vi ritirate non conta, è come se non vi foste neanche iscritti all'appello.
- L'orale facoltativo (che riguarda tutti gli argomenti del corso: teoria, Unix, C) può essere dato in qualsiasi momento dell'anno su appuntamento, ma solo dopo aver discusso l'esercitazione finale e passato lo scritto.

Modalità di Esame

- L'esercitazione di laboratorio verrà presentata intorno a dicembre del 2021, e deve essere sviluppata e discussa entro il *30 NOVEMBRE 2022*, altrimenti occorrerà sviluppare e discutere l'esercitazione successiva, che verrà assegnata intorno a dicembre 2022.

Modalità di Esame

- DURATA: *Il superamento della discussione del laboratorio dà diritto di accedere alla prova scritta per i 5 scritti successivi (orientativamente 1 anno)*.

Modalità di consegna e discussione dell'esercitazione

- Il progetto può essere individuale o di gruppo (massimo tre persone). La discussione del progetto di laboratorio deve essere svolta da tutti i componenti del gruppo contemporaneamente.
- La consegna deve contenere:
  - copia di tutti i sorgenti che costituiscono il progetto;
  - una breve relazione contenente una descrizione degli aspetti caratterizzanti la soluzione stessa (una pagina o due sono normalmente sufficienti); è gradito uno schema grafico a corredo della relazione che illustri la struttura complessiva del sistema realizzato; la relazione deve riportare chiaramente nella prima pagina i nomi dei membri del gruppo, i loro numeri di matricola e gli indirizzi istituzionali di posta elettronica.

Modalità di consegna e discussione dell'esercitazione

- Le consegne di codice e relazione con la richiesta di sostenere la discussione del laboratorio dovranno pervenire almeno 10 giorni prima dalla prova scritta.
- Il voto assegnato al termine della discussione sarà individuale.
  - Nel verificare la soluzione proposta, verranno considerate le competenze e le capacità acquisite dai singoli componenti del gruppo di lavoro (sia relative al linguaggio C e all'interazione con la shell, sia relative agli argomenti studiati nei moduli di laboratorio)
- Per concordare la data della discussione del progetto di laboratorio è necessario inviare l'esercitazione tramite il servizio web indicato sull'esercitazione stessa.

Gruppi

- in generale i gruppi dovrebbero essere formati da studenti dello stesso turno.
- è consentita la realizzazione del progetto di laboratorio da parte di un gruppo composto da studenti di turni diversi alle seguenti condizioni:
  - upload del codice, con indicazione di tutti gli studenti coinvolti;
  - i docenti si consulteranno e comunicheranno al gruppo con quale docente tutti gli studenti del gruppo dovranno effettuare la discussione;
  - tutti gli studenti del gruppo si presenteranno dal docente il giorno prefissato per la discussione, e riceveranno una valutazione individuale.

Plagio

- La consegna di una porzione di codice comporta un'assunzione di responsabilità, in base alla quale chi presenta l'esercitazione afferma di essere autore del codice, e autore di tutto il codice del progetto.
- Non saranno quindi tollerati episodi di plagio, anche parziale, di esercitazioni già consegnate: in questo caso sarà annullato il laboratorio (ed eventualmente, se già verbalizzato, del voto dell'intero corso) *anche agli autori del lavoro*, che si assume siano stati i primi a consegnare e a discutere il progetto.

Registrazione del voto di S.O.

1. Dopo avere superato la discussione del laboratorio, vi iscrivete all'appello scritto e al corrispondente orale verbalizzante, che sono fissati nello stesso giorno.
2. Se superate lo scritto e decidete di accettare il voto conseguito, mandate al Prof. Gunetti una mail in cui richiedete la registrazione del voto.
3. Il voto finale sarà calcolato come la media pesata tra il voto ottenuto allo scritto e il punteggio conseguito nella discussione dell'esercitazione finale.
4. Per la registrazione del voto non è quindi richiesta la vostra presenza, è sufficiente la mail al Prof. Gunetti.

Registrazione del voto di S.O.

- Se vi siete iscritti ad un appello e poi non vi presentate allo scritto, o vi presentate e vi ritirate prima della consegna non succede niente, semplicemente non viene registrato nulla.
- Se invece consegnate lo scritto, quello conta come uno dei tre scritti che potete consegnare in un anno accademico.
- Una volta passato lo scritto, chi ha preso almeno 26 e desidera sostenere l'esame orale facoltativo, prende un appuntamento con il Prof. Gunetti.
- N.B.: i voti registrati in un appello orale verbalizzante compariranno solo qualche giorno dopo la data dell'appello stesso, alla chiusura del registro on-line.

argomenti del laboratorio UNIX

1. introduzione a UNIX;
2. integrazione C: operatori bitwise, precedenze, preprocessore, pacchettizzazione del codice, compilazione condizionale e utility make;
3. controllo dei processi;
4. segnali;
5. pipe e fifo;
6. code di messaggi;
7. memoria condivisa;
8. semafori;
9. introduzione alla programmazione bash.

argomenti trattati oggi: introduzione a UNIX

- UNIX: cenni storici;
- componenti del sistema;
- utenti e account;
- i comandi più importanti;
- proprietà di file e permessi di accesso.

credits: questa lezione è basata su materiale proveniente principalmente da:

Paul Love, Joe Merlino, Craig Zimmerman, Jeremy C. Reed, and Paul Weinstein, *Beginning Unix,* Wiley Publishing, Inc., 2005

cenni storici

un po' di storia

- The Unix operating system was created more than 30 years ago by a group of researchers at AT&T's Bell Laboratories.

  During the three decades of constant development that have followed, Unix has found a home in many places, from mainframes to home computers to the smallest of embedded devices.

un po' di storia

- Unix was developed at AT&T's Bell Laboratories after Bell Labs withdrew from a long-term collaboration with General Electric (G.E.) and MIT to create an operating system called *MULTICS* (Multiplexed Operating and Computing System) for G.E.'s mainframe.

un po' di storia

- Unix was gradually ported to *different machine architectures* from the original PDP-7 minicomputer and was used by universities.
  - The source code was made available at a small fee to encourage its further adoption.
- As Unix gained acceptance by universities, students who used it began graduating and moving into positions where they were responsible for purchasing systems and software.
  - When those people began purchasing systems for their companies, they considered Unix because they were familiar with it, spreading adoption further.

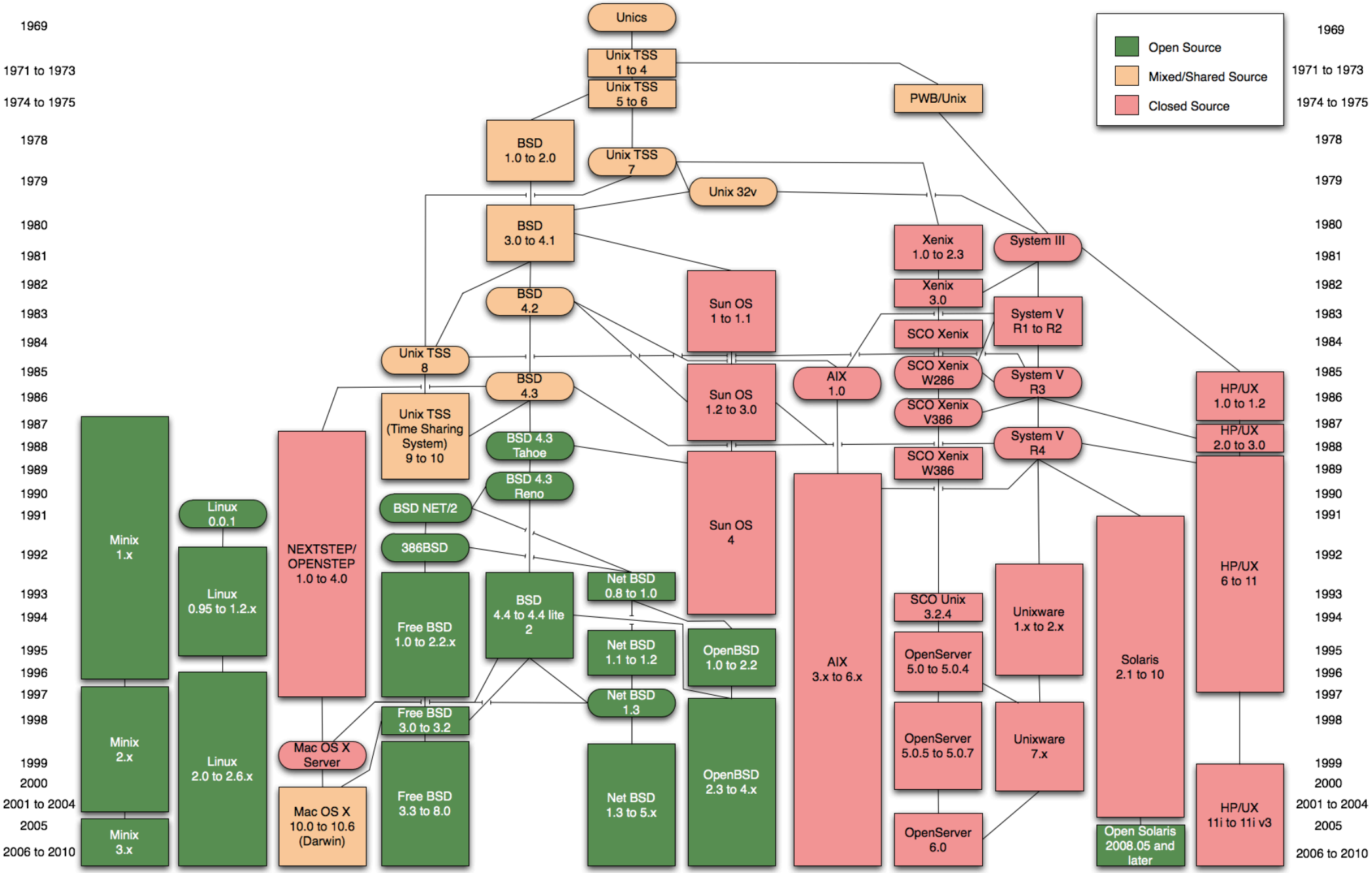
Unix Versions

- There are primarily two base versions of Unix available: AT&T System V and Berkley Software Distribution (BSD).
  - The vast majority of all Unix flavors are built on one of these two versions.

- some of the more popular Unix distributions:

Sun Microsystem's Solaris Unix
IBM AIX
Hewlett Packard HP-UX
Red Hat Enterprise Linux
Fedora Core
SUSE Linux
Debian GNU/Linux
Mac OS X
KNOPPIX
Yellow Dog Linux (for Apple systems)
Santa Cruz Operations SCO
OpenServer
SGI IRIX
FreeBSD
OpenBSD
NetBSD
OS/390 Unix
Plan 9

fonte:
https://www.lifewire.com/unix-flavors-list-4094248



Unix e l'Open Source

- Early Unix systems were mainly commercial commodities like most software for sale.
- In 1983 an engineer named Richard Stallman began work on the GNU Project, which was an effort to create an operating system that was like Unix, and that was open and could be distributed and used freely by anyone.
  - The word "free" in "free software" pertains to *freedom*, not *price*…
- He currently runs the *Free Software Foundation*, and many of the programs he and his supporters have created are used in both commercial and open-source versions of Unix.

http://gnu.org/fsf/fsf.html

Linux

- In 1991 Linus Torvalds, a Finnish graduate student, began work on a Unix-like system called Linux.
  - Linux is actually the kernel, while the parts with which most people are familiar —the tools, shell, and file system — are the creations of others (usually the GNU organization).
- The strength of Linux lies in its progressive licensing, which allows for the software to be freely distributable with no royalty requirements.
  - The only requirement for the end user is that any changes made to the software be made available to others in the community, thus permitting the software to mature at an incredibly fast rate.

componenti

componenti

- An operating system is the software interface between the user and the hardware of a system.
  - Whether your operating system is Unix, DOS, Windows, or OS/2, everything you do as a user or programmer interacts with the hardware in some way.
- the components that make up the Unix operating system are the kernel, the shell, the file system, and the utilities (applications).

Unix Kernel

- The kernel is the lowest layer of the Unix system. It provides the *core capabilities* of the system and allows processes to access the hardware in an orderly manner.
- The kernel controls processes, input/output devices, file system operations, and any other critical functions required by the operating system. It also manages memory.
  - These are all called *autonomous* functions, in that they are run without instructions by a user process.

Unix Kernel

- A kernel is built for the specific hardware on which it is operating, so a kernel built for a *Sun Sparc* machine can't be run on an *Intel* processor machine without modifications.
- Because the kernel deals with very low-level tasks, such as accessing the hard drive or managing multitasking, and is not user friendly, it is generally not accessed by the user.

Unix Kernel - processes

- One of the most important functions of the kernel is to facilitate the creation and management of processes.
- *Processes are executed programs* (called *jobs* or *tasks* in some operating systems) that have owners — human or systems — who initiate their calling or execution.
  - The management of these can be very complicated because one process often calls another (referred to as *forking* in Unix).
- Frequently processes also need to communicate with one another, sending and receiving information that allows other actions to be performed.
  - The kernel manages all of this outside of the user's awareness.

Unix Kernel - memory - vm

- The kernel also manages memory, a key element of any system. It must provide all processes with adequate amounts of memory, and some processes require a lot of it.
- Sometimes a process requires more memory than is available (too many other processes running, for example). This is where *virtual memory* comes in.
  - When there isn't enough physical memory, the system tries to accommodate the process by moving portions of it to the hard disk.
  - When the portion of the process that was moved to hard disk is needed again, it is returned to physical memory. This procedure, called *paging*, allows the system to provide *multitasking* capabilities, even with limited physical memory.

Unix Kernel - memory

- Another aspect of virtual memory is called *swap*, whereby the kernel identifies the least-busy process or a process that does not require immediate execution.
- The kernel then moves the entire process out of RAM to the hard drive until it is needed again, at which point it can be run from the hard drive or from physical RAM.
  - The difference between the two is that paging moves only part of the process to the hard drive, while swapping moves the entire process to hard drive space. The segment of the hard drive used for virtual memory is called the *swap space* in Unix.

Shell

- The shell is a command line interpreter that enables the user to interact with the operating system.
- A shell provides the next layer of functionality for the system; it is what you use directly to administer and run the system.
  - The shell is used almost exclusively via the command line, a text-based mechanism by which the user interacts with the system.
  - There are three major shells available on most systems: the *Bourne shell* (also called sh), the *C shell* (csh), and the *Korn shell* (ksh).

Shell

```
$ cat /etc/shells
# /etc/shells: valid login shells
/bin/csh
/bin/sh
/usr/bin/es
/usr/bin/ksh
/bin/ksh
/usr/bin/rc
/usr/bin/tcsh
/bin/tcsh
/usr/bin/esh
/bin/dash
/bin/bash
/bin/rbash

$ echo $SHELL
/bin/bash
```

File System e Utilities

- The file system enables the user to view, organize, secure, and interact with, in a consistent manner, files and directories located on storage devices.
- Utilities are the applications that enable you to work on the system (not to be confused with the shell).
  - These utilities include the Web browser for navigating the Internet, word processing utilities, e-mail programs, and other commands that will be discussed throughout this course.

Directory	Content
bin/	Common programs, shared by the system, the system administrator and the users.
boot/	The startup files and the kernel.
dev/	Contains references to all the CPU peripheral hardware.
etc/	Most important system configuration files are in /etc, this directory contains data similar to those in the Control Panel in Windows
home/	Home directories of the common users.
lib/	Library files, includes files for all kinds of programs needed by the system and the users.
lost+found/	Every partition has a lost+found in its upper directory. Files that were saved during failures are here.
mnt/	Standard mount point for external file systems, e.g. a CD-ROM or a digital camera.
opt/	Typically contains extra and third party software.
root/	The administrative user's home directory. Mind the difference between /, the root directory and /root, the home directory of the root user.
sbin/	Programs for use by the system and the system administrator.
tmp/	Temporary space for use by the system, cleaned upon reboot, so don't use this for saving anything
usr/	Programs, libraries, documentation etc. for all user-related programs.
var/	Storage for all variable files and temporary files created by users, such as log files, the mail queue, the print spooler area, space for temporary storage of files downloaded from the Internet, or to keep an image of a CD before burning it.

filosofia

filosofia Unix (1)

- Simplicity: Many of the most useful UNIX utilities are very simple and, as a result, small and easy to understand.
  - "Small and Simple" is a good technique to learn.
- Focus: It's often better to make a program perform one task well than to throw in every feature along with the kitchen sink.
  - Programs with a single purpose are easier to improve as better algorithms or interfaces are developed.

filosofia Unix (2)

- Reusable Components: Make the core of your application available as a library. Well-documented libraries with simple but flexible programming interfaces can help others to develop variations or apply the techniques to new application areas.
- Filters: Many UNIX applications can be used as filters. That is, they transform their input and produce output.
  - UNIX provides facilities that allow quite complex applications to be developed from other UNIX programs by combining them in novel ways.

filosofia Unix (3)

- Open File Formats: The more successful and popular UNIX programs use configuration files and data files that are plain ASCII text.
  - It enables users to use standard tools to change and search for configuration items and to develop new tools for performing new functions on the data files.
- Flexibility: You can't anticipate exactly how ingeniously users will use your program.
  - Try to be as flexible as possible in your programming.
  - Try to avoid arbitrary limits on field sizes or number of records.
  - Never assume that you know everything that the user might want to do.

Accounts

cosa sono gli account

- A user account provides you with access to the Unix system, whether by a *shell*, an *ftp* account, or other means.
  - To use the resources that the Unix system provides, you need a valid *user account* and resource *permissions*.

Tipi di account

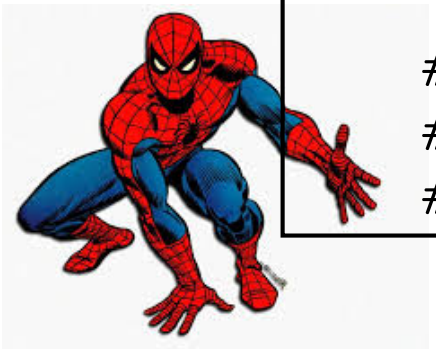
- There are three primary types of accounts on a Unix system:
  - the root user (or superuser) account,
  - system accounts, and
  - user accounts.
- Almost all accounts fall into one of those categories.

Root

- The root account's user has complete control of the system, to the point that he can run commands to completely destroy the system.
- The root user (also called root) can do absolutely anything on the system, with no restrictions on files that can be accessed, removed, and modified.

```
We trust you have received the usual lecture from the
local System Administrator. It usually boils down to
these three things:

    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.
```



Root

- The Unix methodology assumes that root users know what they want to do, so if they issue a command that will completely destroy the system, Unix allows it.
- This basic tenet is why people generally use root for only the most important tasks, and then use it only for the time required — and very cautiously.

System Accounts

- System accounts are those needed for the operation of system-specific components. They include, for example, the *mail* account (for electronic mail functions) and the *sshd* account (for *ssh* functionality).
  - They generally assist in the running of services or programs that the users require.
- some of them may not exist on your Unix system. These accounts are usually needed for some specific function on your system, and any modifications to them could adversely affect the system.
  - some of the system account names you may find in your */etc/passwd* file are adm, alias, apache, backup, bin, bind, daemon, ftp, guest, gdm, gopher, mail, mysql, etc.

System Accounts

```
$ cat /etc/passwd
##
# User Database
#
# Note that this file is consulted directly only when the system is
running
# in single-user mode.  At other times this information is provided by
# Open Directory.
#
# See the opendirectoryd(8) man page for additional information about
# Open Directory.
##
nobody:*:-2:-2:Unprivileged User:/var/empty:/usr/bin/false
root:*:0:0:System Administrator:/var/root:/bin/sh
daemon:*:1:1:System Services:/var/root:/usr/bin/false
_uucp:*:4:4:Unix to Unix Copy Protocol:/var/spool/uucp:/usr/sbin/uucico
_taskgated:*:13:13:Task Gate Daemon:/var/empty:/usr/bin/false
_networkd:*:24:24:Network Services:/var/empty:/usr/bin/false
_installassistant:*:25:25:Install Assistant:/var/empty:/usr/bin/false
...
```

User Accounts

- User accounts provide interactive access to the system for users and groups of users.
- General users are typically assigned to these accounts and usually have limited access to critical system files and directories.
  - Generally you want to use eight characters or fewer in an account name, but this is no longer a requirement for all Unix systems.
  - For interoperability with other Unix systems and services, however, you will most likely want to restrict your account names to eight characters or fewer.

Group Accounts

- Group accounts add the capability to assemble other accounts into logical arrangements for simplification of privilege (permission) management.

Group Accounts

- Unix permissions are placed on files and directories and are granted in three subsets:
  - the owner of the file, also known as the *user*;
  - the group assigned to the file, also known simply as *group*; and
  - anyone who has a valid login to the system but does not fall into either the owner or group subsets, also known as others.
- The existence of a group enables a resource or file owner to grant access to files to a class of people.

Esempio

- For example, say that a company with about 100 employees uses a central Unix server.
- Three of the employees compose the company's human resources (HR) staff; they often deal with sensitive information, including salaries, pay raises, and disciplinary actions. The HR staff has to store its information on the server everyone else uses, but its directory, Human_Resources, needs to be protected so that others cannot view the contents.
- To enable HR to set specific permissions on its files that allow access only to HR staff, the three staff members are put into a group called *hr*. The permissions on the Human_Resources directory can then be set to allow those members to view and modify files, while excluding all who fall into the other group (everyone else).

Gruppi

- One of the strengths of groups is that an account can belong to many groups, based on access requirements.
  - For instance, the two members of the internal audit team may need to access everyone's data, but their directory, called Audit, needs to be protected from everyone else's account.
  - To do this, they can belong to all groups and still have a special audit group in which they are the only members.

File Ownership and Permissions

A multiuser system

- One of the distinguishing features of Unix is that it was designed from its earliest days to be a multiuser system.
- Because of its multiple-user design, Unix must use mechanisms that enable users to manage their own files without having access to the files of other users.
- These mechanisms are called file ownership and file permissions.

File Ownership

- Generally, the files that the user owns are ones that he created, or which were created as a result of some action on his part.
- The exception to this, of course, is the *superuser,* also known as *root*.
  - The superuser can change the ownership of *any* file, whether he created it or not, with the *chown* command. For example, if the superuser gives the command

```
chown jane /home/bill/billsfile
```

  - the ownership of the file */home/bill/billsfile* is transferred to the user jane.

File Permissions

- As the owner of a file, a user has a right to decide who can access a file, and what kind of access others can have.
- There are three kinds of file permission:
  - [r] Read (file can be viewed)
  - [w] Write (file can be edited)
  - [e] Execute (file can be run as a program)
- there are three categories of users to whom these permissions can be applied:
  - [u] User (owner of that particular file)
  - [g] Group (group to which the file is assigned)
  - [o] Others (all users and groups)

File Permissions

- When you define permissions for a given file, you must assign a type of permission to each category (**`u,g,o`**) of users.
  - If you are writing a program, you probably want to give yourself *read*, *write*, and *execute* permission.
  - If you are working with a team, you might want to have your system administrator create a group for the team.
    - You can then give the team read and execute permissions so they can test your program and make suggestions but not be able to edit the file.

How to Read a Permissions List

- use the *ls -l* command, and the permission information is printed as part of a directory listing.
- from left to right, the columns show file permissions, UID, username, group name, file size, timestamp, and filename

```
drwxr-xr-x 1 radicion  staff  272 Sep 20 17:40 01_intro_UNIX.key
drwxr-xr-x 1 radicion  staff  306 Sep 20  2021 lezione_01_esercizi
-rw-r--r-- 1 radicion  staff   38 Jan 12  2020 proxy.txt
```

Permissions

- The string of characters on the left displays all the permission information for each file.
- The permission information consists of nine characters, beginning with the second character from the left.
  - The first three characters represent the permission for the user, the second three for the group, and the final set of three represents permission for all.

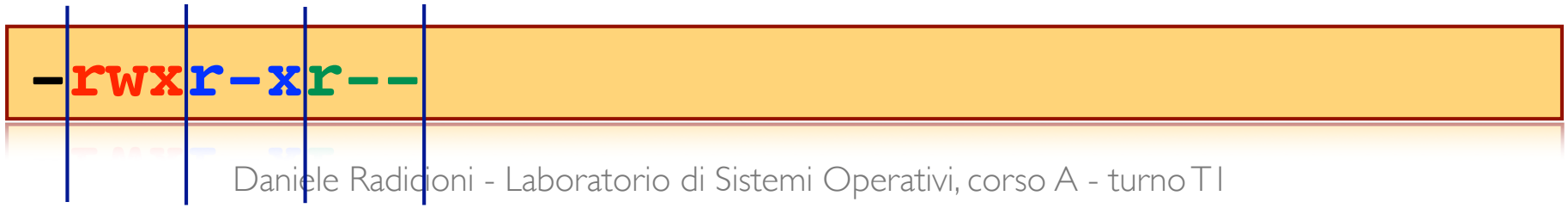
Permissions

Permission	*Applied to a Directory*	*Applied to Any Other Type of File*
`read (r)`	Grants the capability to read the contents of the directory or subdirectories.	Grants the capability to view the file.
`write (w)`	Grants the capability to create, modify, or remove files or subdirectories.	Grants write permissions, allowing an authorized entity to modify the file, such as by adding text to a text file, or deleting the file.
`execute (x)`	Grants the capability to enter the directory.	Allows the user to "run" the program.
-	No permission.	No permission.

```
-rwxrwx---
```

Permissions

Characters	*Apply to*	*Definition*
rwx (characters 2–4)	The owner (known as user in Unix) of the file	The owner of the file has read (or view), write, and execute permission to the file.
r-x (characters 5–7)	The group to which the file belongs	The users in the owning group (users) can read the file and execute the file if it has executable components (commands, and so forth). The group does not have write permission: the - character fills the space of a denied permission.
r-- (characters 8–10)	Everyone else (others)	Anyone else with a valid login to the system can only read the file: write and execute permissions are denied (--).



Changing Permissions

- To change file or directory permissions, you use the *chmod* (change mode) command. There are two ways to use *chmod*: symbolic mode and absolute mode.
- In symbolic mode, **+** (**–**) adds (removes) the designated permission(s) to a file or directory

```
chmod o+wx myfile
```

  - Adds write and execute permissions for others

- In Absolute mode

```
chmod 754 myfile
```

111	101	100
rwx	**r-x**	**r--**
7	5	4

```
ls -l file1
-rw-r--r-- 2 radicion radicion    6 2020-09-03 13:56 file1

chmod 700 file1
ls -l file1
-rwx------ 2 radicion radicion    6 2020-09-03 13:56 file1

chmod o+wx file1
ls -l file1
-rwx----wx 2 radicion radicion    6 2020-09-03 13:56 file1

chmod o-w file1
ls -l file1
-rwx-----x 2 radicion radicion    6 2020-09-03 13:56 file1
```

Comandi

Commands

- Since its earliest days, Unix has primarily used a command-line interface: a simple prompt, followed by a cursor, using only text.
- In this kind of environment, there is only one mode of interacting with the machine, and that's via commands.
- Commands are executable programs.
  - Sometimes they are stand-alone programs, and
  - sometimes they are functions that are built into the shell. In either case, a command enables the user to request some sort of behavior from the machine.

Anatomy of a Command

- A Unix command can be broken down into two parts: the command itself and the arguments appended to it.
  - For example, the *ls* command is used to list the contents of a directory. If you type *ls* at the command prompt, the contents of the working directory (that is, the directory in which you are currently located) is printed to the screen

```
$ ls
Applications   Documents Library    Music    Public      Sites
VirtualBox VMs Desktop   Downloads Movies    Pictures
```

Anatomy of a Command

- With the use of arguments, you can influence the behavior or output of the command.
  - An argument adds additional information that changes the way in which the command executes.

```
$ ls /etc
6to4.conf          hosts.equiv          php.ini-5.2-previous
AFP.conf           irbrc                php.ini.default
afpovertcp.cfg     kern_loader.conf     ...
```

Anatomy of a Command

- Arguments can influence the form of a command's output as well as its operation.
- the *ls* command would show the directory listing for */etc* in long format, which provides additional information about the files being listed.

```
$ ls -l /etc
total 1780
-rw-r--r--   1 root    root    15221  Feb 28 2001 a2ps.cfg
-rw-r--r--   1 root    root     2561  Feb 28 2001 a2ps-site.cfg
-rw-r--r--   1 root    root       47  Dec 28 2001 adjtime
...
```

Arguments and Targets

- Flags are usually —but not always— preceded by a dash, or hyphen.
  - The *tar* command, for example, takes its flags without a prepended dash.
- In addition to accepting multiple arguments at one time, some commands require multiple targets (e.g. *cp*).

```
$ tar cvzf foo.tgz cps100
$ tar xvzf foo.tgz

$ cp prog.c prog.bak
```

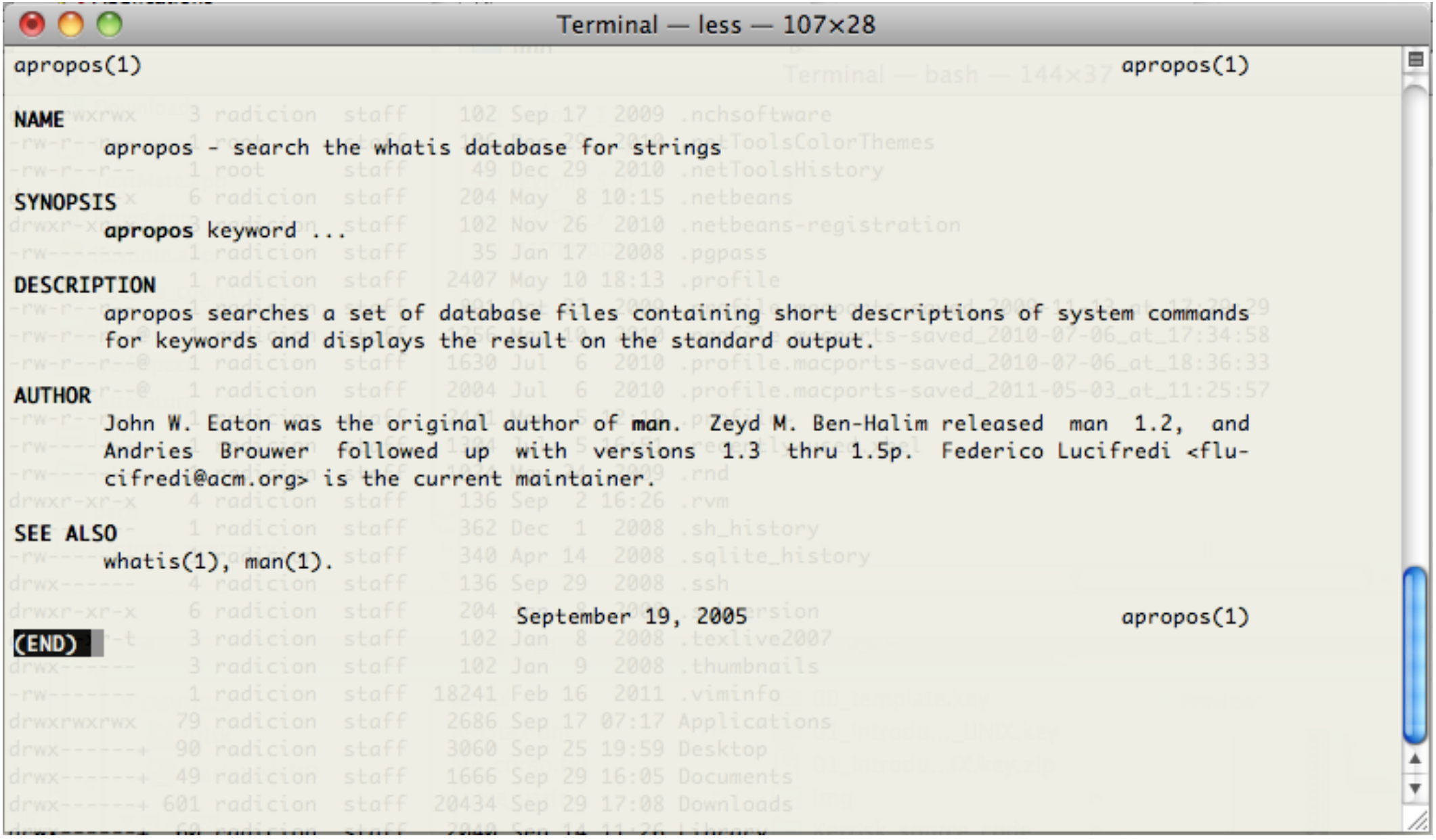
Finding Information on Commands

- Unix manual pages (or man pages) are the best way to learn about any given command. Find a particular manual page with the command *man command*.

Section	Description
1	General commands
2	System calls
3	Library functions, covering in particular the C standard library
...	

```
$ man 1 printf
$
$ man 3 printf
```

Finding Information on Commands



Command Modification

- Unix commands are not limited to the functions performed when the command name is typed at the prompt.
- You can use a number of different tools to enhance or alter the function of a command, or to manage the command's output.

Metacharacters

- One of the more interesting aspects of using a command-line interface is the capability to use special characters called *metacharacters* to alter a command's behavior.
- wildcards are special characters that can be used to match multiple files at the same time
  - `?` matches any one character in a filename.
  - `*` matches any character or characters in a filename.
  - `[ ]` matches one of the characters included inside the [ ] symbols.

Metacharacters and *ls* command

- suppose that the working directory contains the files:

  `date help1 help2 help3 myprog.f myprog.o`

- You can add wildcards to the target argument of the *ls* command to find any or all of these files in a single search.

Argument + Wildcard	*Files Matched*
`help?`	help1 help2 help3
`myprog.[fo]`	myprog.f myprog.o
`*`	date help1 help2 help3 myprog.f myprog.o
`*.f`	myprog.f
`help*`	help1 help2 help3

Input and Output Redirection

- To function, every command needs a source of input and a destination for output.
  - In the vast majority of cases, the standard input is the keyboard, and the standard output is the screen — specifically, the *Terminal* window, if you're using a graphical interface.

- Redirection allows to force a particular command to take input from a source other than the keyboard, or to put the output somewhere besides the monitor.
  - For example, you might want to set up a program to run without having to be at the keys to invoke it, and then to dump its output into a text file

Input and Output Redirection

- Input and output redirection uses the < and > characters to define the temporary input and output sources.
  - Suppose that you want to use *ls* to find the contents of a directory, but you want the output captured in a text file rather than printing to the screen. To do so, create a file named *ls_output* and then issue the command:

```
$ ls > ls_output
```

- The > character takes the output of *ls*, which would normally go to the screen, and writes it to the *ls_output* file.

Input and Output Redirection

- If the specified file does not already exist, the > operator will create it.
- If the preceding command were issued twice, the second command would overwrite the contents of the *ls_output* file and destroy the previous data.
  - If you want to preserve existing data, use the >> operator, which appends the new data to the end of the file. If the specified file doesn't exist, or is empty, >> acts just like >.

Input and Output Redirection

- In the same way that > redirects the output of a command, < can be used to change the input.

- For example, assume that you want to alphabetize a list of terms contained in a file called 'terms'. You can use the sort command in combination with the input redirection operator <, as in:

```
sort < terms
```

- Input and output redirection can also be combined. For example, the following command will sort the items in the terms file and then send the output of the sort into a new file called terms-alpha.

```
sort < terms > terms-alpha
```

Pipes

- A pipe is an operator that combines input and output redirection so that the output of one command is immediately used as the input for another. the pipe is represented by the vertical line character: '|'
  - Suppose you want to list the contents of a long directory. A pipe gives you a simple way to display the output one page at a time, with the following command:

```
$ ls -l /etc | more
total 1780
-rw-r--r--  1 root   root   15221  Feb 28 2001 a2ps.cfg
-rw-r--r--  1 root   root    2561  Feb 28 2001 a2ps-site.cfg
-rw-r--r--  1 root   root      47  Dec 28 2001 adjtime
...
```

Navigating the File System

Command	*Description*
`cat`	Concatenate: displays a file.
`cd`	Change directory: moves you to the directory identified
`cp`	Copy: copies one file/directory to specified location.
`file`	Identifies the file type (binary, text, etc).
`find`	Finds a file/directory.
`head`	Shows the beginning of a file.
`less`	Browses through a file from end or beginning.
`ls`	List: shows the contents of the directory specified.
`mkdir`	Make directory: creates the specified directory.
`more`	Browses through a file from beginning to end.
`mv`	Move: moves the location of or renames a file/directory.
`pwd`	Print working directory: shows the current directory the user is in.
`rm`	Remove: removes a file.
`tail`	Shows the end of a file.
`touch`	Creates a blank file or modifies an existing file's attributes.
`which`	Shows the location of a file if it is in your PATH.

ls

- The most common Unix commands are those used to manage files and directories. The *ls* command takes the syntax:

```
ls [options] [directory]
```

Argument	*Function*
`-l`	Lists directory contents in long format, which shows individual file size, permissions, and other data.
`-t`	Lists directory contents sorted by the timestamp (time of the last modification).
`-a`	Lists all directory contents, including hidden files whose name begins with the . character.
`-i`	Lists directory contents including inode or disk index number.
`-R`	Lists directory contents including all subdirectories and their contents.

```
$ ls -l /etc
drwxr-xr-x   2 root root  4096 2020-09-12 09:08 bin
```

Argument	*Function*
`drwxr-xr-x`	The type of file and the permissions associated with it
`root`	The owner of the file
`root`	The group to which the file owner belongs
`4096`	Size of file (in characters)
`2020-09-12 09:08`	The last time the file or directory was modified
`bin`	Name of file or directory

cd

- The *cd* command is something that you will use frequently. It's the command used to move through the file system. It takes the syntax:

```
cd directory

cd .. // to move one level up on the hierarchy

cd    // to move to the /home dir
```

cat

- If you want to see the contents of a given file, there's no easier way than the *cat* command, which prints the contents of a specified file to the standard output, usually the monitor. It takes the following syntax:

```
cat [options] file(s)
```

  - *-n* Numbers the output's lines;
  - *-v* Shows all nonprinting characters

- to concatenate multiple files into one larger new file,

```
cat file1 file2 file3 >> newfile
```

more/less

- Virtually identical commands, used to break up command output or file contents into *single-screen chunks* so that you can read the contents more easily. Both *more* and *less* can move forward through a file, although only *less* can be used to move backward.

```
more filename
```

```
less filename
```

- Press the spacebar to advance to the next screen. If you are using *less* to view the output or file, you can use the *B* key to move back one screen. To return to the command prompt press *Q* in either *more* or *less*.

mv

- The mv command is used to move a file from one location to another. It takes the syntax:

```
mv old new
```

- where *old* is the current name of the file to be moved to the location defined as *new*.
  - If the value of *new* is simply a new filename, the file is renamed and remains in the current directory.
  - If the value of *new* is a new directory location, the file is moved to the new location with the existing filename.
  - If the value of *old* is a directory, the entire directory and its contents will be moved to the location specified as *new*.

cp

- Like the mv command, *cp* is used to create new files or move the content of files to another location. Unlike *mv*, however, *cp* leaves the original file intact at its original location. *cp* uses the syntax

```
cp file1 file2
```

- where *file1* is the original file and *file2* is the destination file.
  - If you use the name of an existing file as the destination value, *cp* overwrites that file's contents with the contents of *file1*.

rm

- The rm command is used to delete a file. It uses the syntax:

  `rm [options] filename`

- This command can be quite destructive unless you are careful when you issue it, especially if you use wildcards.
  - For example, the command *rm conf** deletes all files beginning with the characters *conf*, whether you wanted to delete those files or not.

Argument	*Function*
`-i`	Forces interactive mode, prompting you to confirm each deletion
`-r`	Forces recursive mode, deleting all subdirectories and the files they contain
`-f`	Forces force mode, ignoring all warnings (very dangerous)

grep

- stands for "global regular expression print," processes text line by line and prints any lines which match a specified

```
pattern.rm [options] filename
```

Argument	*Function*
`-i`	to ignore word case i.e match pippo, Poo, PIPPO and all other combination
`-r`	search recursively i.e. read all files under each directory for a string
`-n`	to precede each line of output with the number of the line in the text file from which it was obtained
`-v`	it matches only those lines that do not contain the given word