Some LINUX commands

CS 2XA3

Term I, 2018/19

Outline

Man pages

If you do not know how to use a command *cmd*, type

```
man cmd
```

For instance:

```
man grep
man ls
```

Man pages

`apropos` *keyword ...*

will list man entries containing one of the keywords

For instance:

```
apropos open file
apropos list directory
```

Listing files

`pwd`	show current directory
`ls`	list files in current directory
`ls -a`	show hidden files, e.g. `.xxx`
`ls -l`	output in long format
`ls -lt`	sort by time, most recent at the top
`ls -lrt`	sort by time, most recent at the bottom
`ls -lS`	sort by size, largest at the top
`ls -rlS`	sort by size, largest at the bottom
`ls -F`	denote directories with `/`

Showing file content

`cat file`	output content of the ***file***
`more file`	similar
`less file`	similar
`tail file`	just the end of the file

File commands

`cp old new`	copies ***old*** to ***new***
danger - if *new* exists, it is overridden without warning	
`cp -i old new`	like `cp` but prompts for confirmation
`mv old new`	moves (renames+relocates) ***old*** to ***new***
danger - if *new* exists, it will be replaced without warning	
`mv -i old new`	like `mv` but prompts for confirmation
`rm file`	deletes ***file***
danger, *file* will be deleted without warning	
`mkdir dir`	creates directory ***dir***
`rmdir dir`	removes directory ***dir***, but only if it is empty

File commands

All commands except `mkdir` work recursively for subdirectories if option `-r` or `-R` is used.

- `rm -r dir` will remove directory *dir* even if it is not empty and without warning !!
- `rm *` will delete all files without warning !!
- `rm -r *` will delete everything without warning !!

There is no undelete in UNIX
Deleted files cannot be recovered

Finding files

`locate file`	finds where *file* is located
`which cmd`	tells where *cmd* is located

For instance:

```
which bash
```

Wildcards

`*` matches 0 or more characters

e.g. `ls *` will list all files

`?` matches a single character

e.g `ls file?` will list `file1`, `fileA`, etc., but not `file11`

`[...]` will match any combination of any length consisting of the letters in `[ ]`

e.g. `file[1..9]` will match `file1`, ... `file9`, ...

`file11`, ..., ..., `file99`, ..., etc.

File permissions

Examples

```
drwxr-xr-x 8 franek faculty 4096 Aug 16 01:36 MyDir
```

directory, first symbol is **d**

```
-rwxr-xr-x 8 franek faculty 4096 Aug 16 01:36 MyFil
```

regular file, first symbol is **-**

first symbol	meaning
-	regular file
d	directory
b	block device, e.g. hard disk, CD-ROM
c	character device, e.g. mouse
s	socket device (inter-process communication)

File permissions

symbol	meaning
`r`	can read
`w`	can write, create
`x`	can execute, search
`-`	cannot

- $\underbrace{r\ w\ x}_{\text{user}}\ \underbrace{r\ w\ x}_{\text{group}}\ \underbrace{r\ w\ x}_{\text{others}}$

```
-rwxr-xr-x 8 franek faculty 4096 Aug 16 01:36 MyFil
```

- it is a regular file
- franek can read, write, execute the file `MyFil`
- members of the group faculty can read, execute, cannot write
- all others can read, execute, cannot write

File permissions

```
drwxr-xr-x 8 franek faculty 4096 Aug 16 01:36 MyDir
```

- it is a directory
- franek can read which files are in the directory, can create/modify/delete files in the directory, can search the directory
- members of the group faculty can read files in the directory, can search the directory, cannot create/modify/delete files in the directory
- all others can read files in the directory, can search the directory, cannot create/modify/delete files in the directory

Changing file permissions

```
chmod [-R] who op perm file(s)
```

`who`	user, group, others, all
`op`	+ add permission, − remove permission
`perm`	read, write, execute

Examples

```
chmod a+rwx XXX
chmod og-rwx XXX

-rwx------
```

grep

`grep regex file`

for us mostly in a simplified version

`grep string file`

searches for `string` in the file `file`
shows the lines containing the string `string`

`grep -v string file`

searches for `string` in the file `file`
shows the lines that do not contain the string `string`

grep examples

```
grep main *.c
```

searches all C programs in directory and display the lines containing the word `main`

```
grep '2\|3' *
```

searches all files in directory looking for characters `2` or `3`

Input/Output

Most commands take input from ***standard input*** – normally keyboard – and output the result into ***standard output*** – normally screen.

`< file(s)` redirects standard input to the file (or files) `file(s)` and the command or program reads from there instead from the keyboard

Input/Output

`tr 'a' 'b'`

reads what you type and translates every `a` to `b` and displays the output on the screen (line by line)

`tr 'a' 'b' < XXX`

reads file XXX and translates every `a` to `b` and displays the output on the screen (line by line)

Input/Output

`> file`

redirects standard output to the file `file` creating the file or rewriting its contents (if it existed)

`>> file`

redirects standard output to the file `file` creating the file (if it did not exist) or appending it to its contents (if it existed)

Input/Output

```
tr 'a' 'b' < xxx > yyy
```

reads file `xxx` and translates every `a` to `b` and writes the output into the file `yyy`, previous content of `yyy` overridden if any

```
tr 'a' 'b' < xxx » yyy
```

reads file `xxx` and translates every `a` to `b` and appends the output to the file `yyy`

Input/Output

```
echo "hello"
```

will display `hello` on the screen

```
echo "hello" > xxx
```

will create a file `xxx` with one line: `hello` or will override an existing `xxx` with one line: `hello`

Input/Output

```
echo "hello" > XXX
echo "Peter" > XXX
```

will create a file `XXX` with one line: `Peter` or will override an existing `XXX` with one line: `Peter`

```
echo "hello" > XXX
echo "Peter" » XXX
```

will create a file `XXX` with two lines: `hello` and `Peter` or will override an existing `XXX` with two lines: `hello` and `Peter`

Input/Output - pipes

the output of one command can be the input for another command:

```
who | grep franek
```

`who` will output a list of all current users, `grep` will read it looking for a line containing `franek`

Input/Output - pipes

```
who
```

```
zucker pts/0 2015-09-02 00:59 (d24-141-99-232.home.cgocable.net)
zucker pts/1 2015-09-02 16:15 (jzmac.cas.mcmaster.ca)
franek pts/2 2015-09-03 05:02 (aputeaux-554-1-28-104.w86-249.abo.wanadoo.fr)
zucker pts/3 2015-09-02 00:59 (d24-141-99-232.home.cgocable.net)
zucker pts/4 2015-08-23 12:35 (jzmac.cas.mcmaster.ca)
zucker pts/9 2015-08-23 12:35 (jzmac.cas.mcmaster.ca)
janicki pts/10 2015-09-02 11:37 (bas8-hamilton14-3096449871.dsl.bell.ca)
```

```
who | grep franek
```

```
franek pts/2 2015-09-03 05:02 (aputeaux-554-1-28-104.w86-249.abo.wanadoo.fr)
```

Execution control

`ps`	lists all processes
`ps - franek`	lists all processes of user `franek`
`kill -sigl pid`	sends a signal `sig` to the processes with the process id `pid`
`kill -9 pid`	terminates the processes with the process id `pid`
`&`	to run a program in background e.g. `prog1 &`
`fg`	to bring a process to foreground or to revive a suspended program
`CTRL-Z`	suspends a program
`CTRL-C`	stops a program