CS6 Practical System Skills

Fall 2019 edition

Leonhard Spiegelberg
lspiegel@cs.brown.edu

02.99 Recap

Last lecture:

- file paths (absolute / relative)
- commands, man pages
- ls, cd, pwd

Today:

- creating, managing and accessing files
- UNIX wildcards
- UNIX users and file permissions

02.99 Recap

file separator `/`

absolute path ⇒ starts with `/`

relative path ⇒ path with special symbols `.` `..` `~`

change directory `cd`

list `ls`

02.99 Recap quiz

Assuming we are in `/usr/local/bin`, where would the following commands take us?

```
cd ..

cd ./../..

cd

cd ~
```

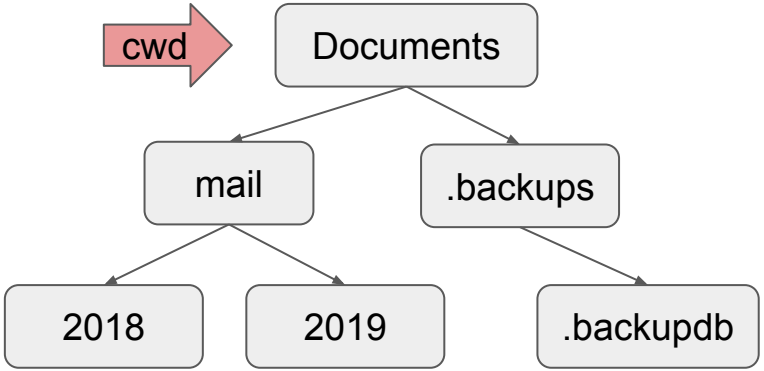
02.99 Recap quiz

Assuming we are in `/usr/local/bin`, where would the following commands take us?

`cd ..`	`/usr/local`
`cd ./../..`	`/usr`
`cd`	`HOME directory`
`cd ~`	`HOME directory`

02.99 Recap quiz

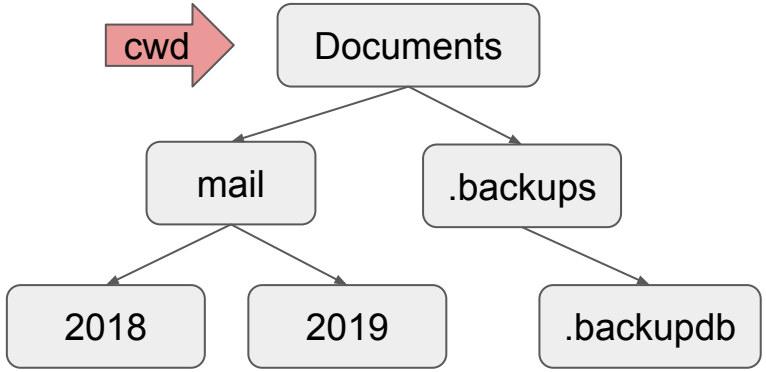
Assuming we have the following file tree structure, which files would be listed?




```
ls Documents

ls

ls -a
```

03.00 Recap quiz

`ls Documents`
`ls: Documents: No such file or directory`
`ls`
`mail`
`ls -a`
`. .. mail .backups`



03 Files

CS6 Practical System Skills

Fall 2019

Leonhard Spiegelberg *lspiegel@cs.brown.edu*

03.01 Creating and deleting files

files		directories	
`touch file`	create a new file if it doesn't exist, update access/modification time	`mkdir directory_name`	create a new directory, fails if directory already exists
`rm file`	remove file	`rmdir directory_name`	remove directory, if it is **empty**

03.02 Working with directories

create recursive directories on the fly

`-p` : create intermediate directories as required

Example:

```
mkdir -p folder/subfolder/subsubfolder
```

03.03 Deleting files and directories

WARNING

In UNIX, there is NO WAY to restore files/directories once you delete them using rmdir/rm. Understand and be very careful with these commands.

03.03 Deleting files

delete directories, subdirectories and files

`-r` : recursively remove files in hierarchy

```
rm -r folder/
```

03.03 Deleting files in a tree

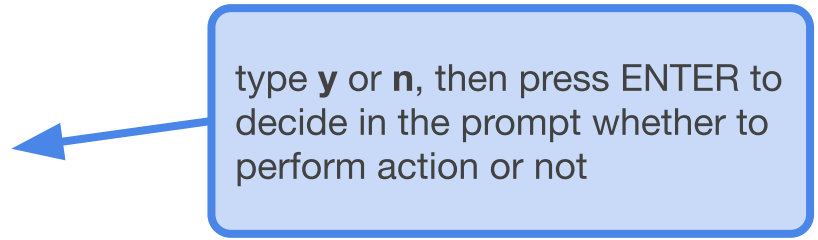
delete directories, including all the files and subdirectories in them

`-r` recursively remove files in hierarchy

`-i` interactive removal mode, asks for permission before each file is removed

`Example:`

```
rm -ri folder/
```



03.03 Deleting files in a tree

delete directories, including all files and subdirectories in them

`-r` recursively remove files in hierarchy

`-f` force removal, silent prompts & warnings

Example:

```
rm -rf folder/
```

Back to the terminal...

03.04 Organizing files

works for directories too

`mv src ... dest`	move file from src location to dest location ⇒ can be used to rename a file
`cp src ... dest`	copy **file** from src location to dest location

works for **files** only

03.04 Copying directories

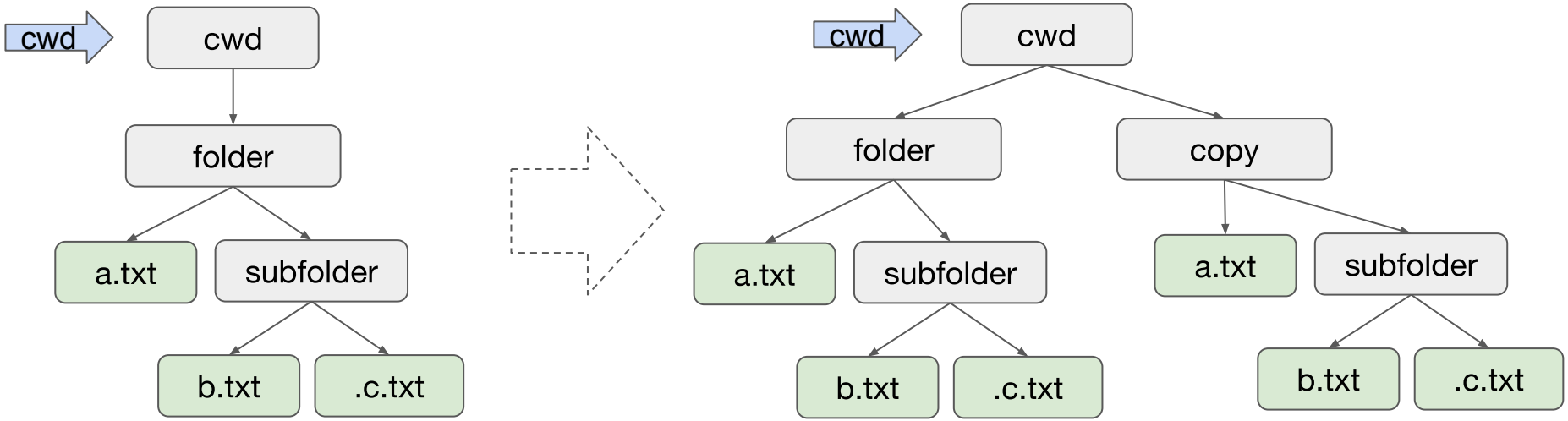
`cp -R src dest`

Two special cases:

1.) if src designates a directory, cp copies the directory and the entire subtree
2.) if src ends with a `/` (trailing slash!), `cp` copies the contents of the directory recursively, but not the directory itself if dest exists. (Else, it defaults to 1.))

03.04 Copying directories

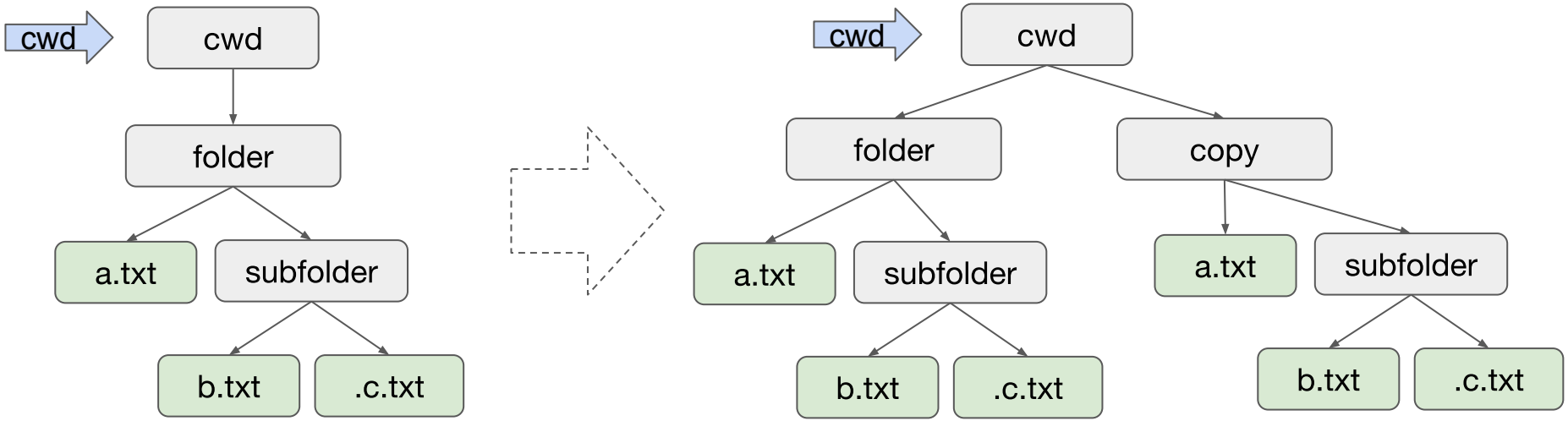
```
Examples: cp -R folder copy
```



03.04 Copying directories

```
Examples: cp -R folder/ copy
```

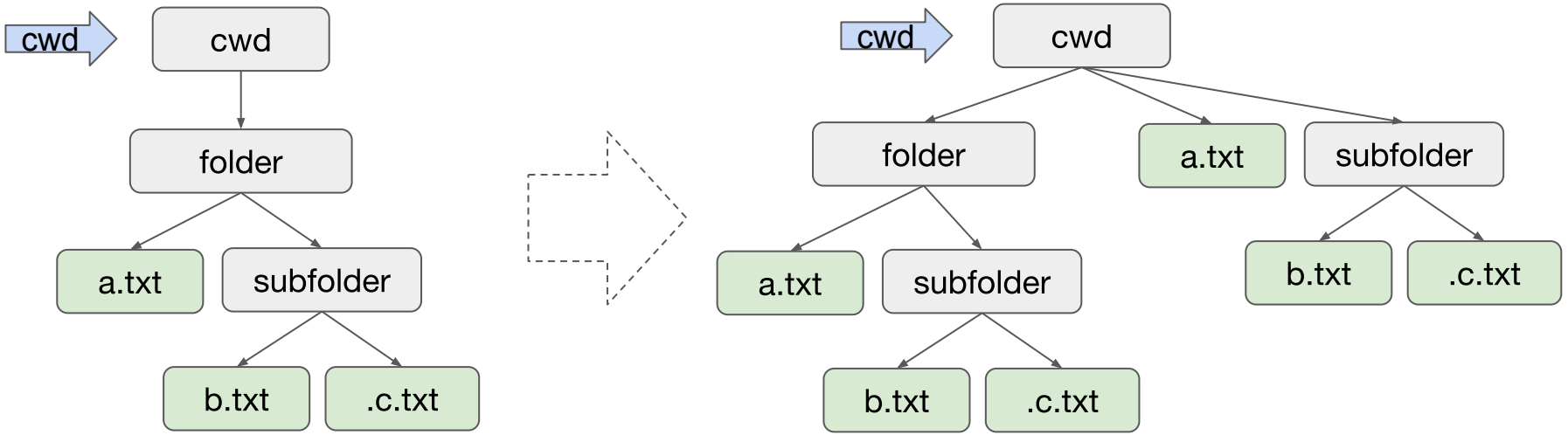
trailing / has no effect, because copy does not exist yet



03.04 Copying directories

```
Examples: cp -R folder/ .
```

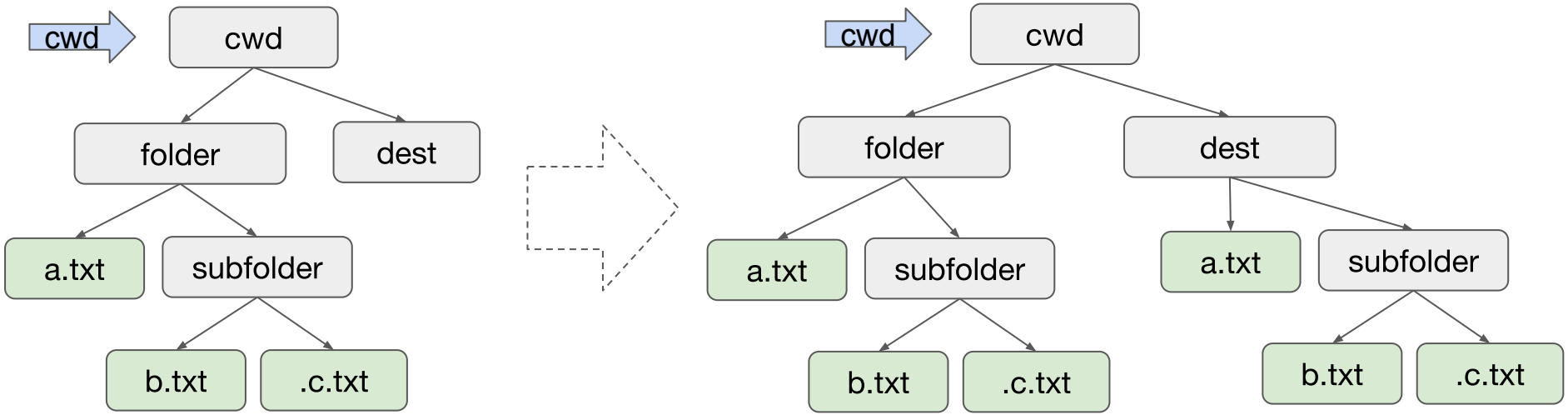
. exists, so its contents are copied!



03.04 Copying directories

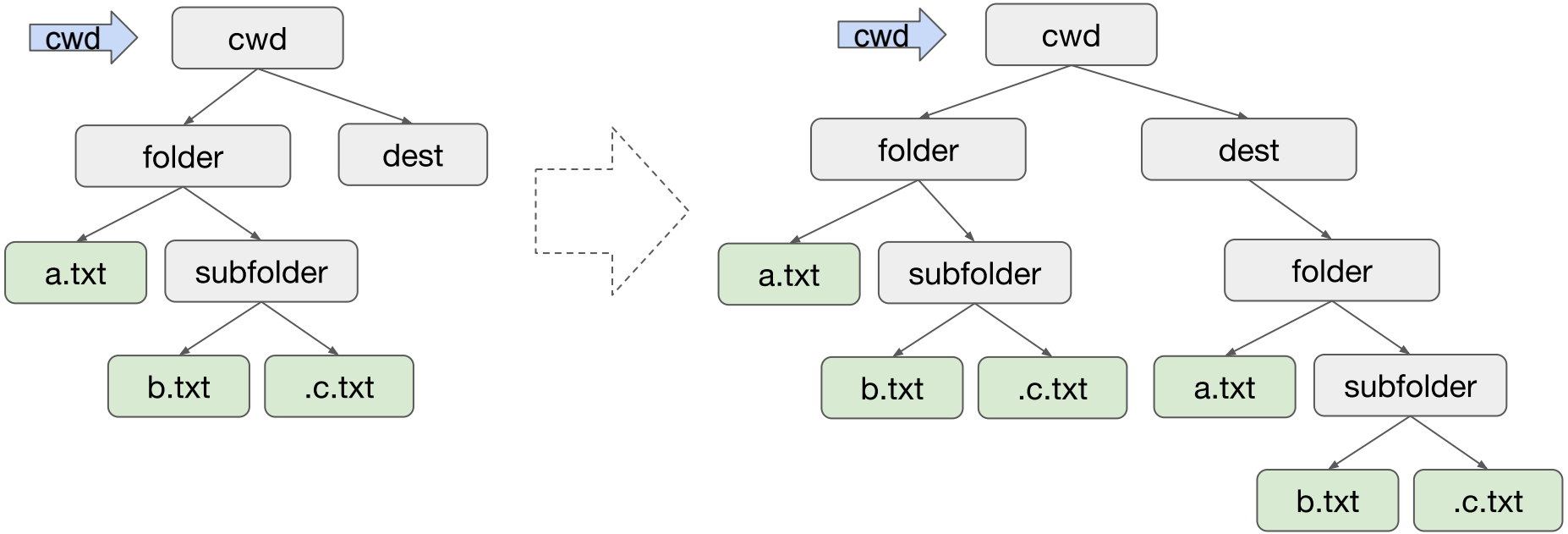
```
Examples: cp -R folder/ dest
```

dest exists, so its contents are copied!



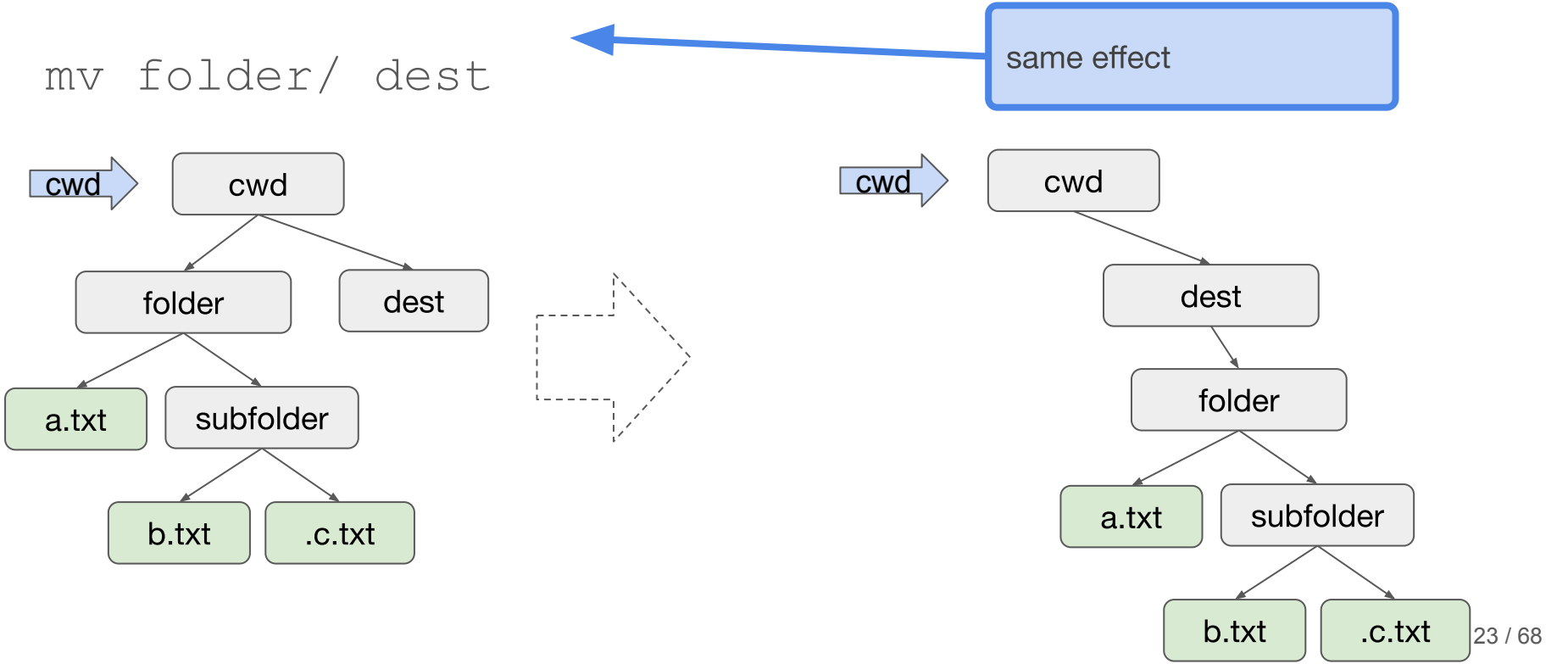
03.04 Copying directories

```
Examples: cp -R folder dest
```



03.04 Moving directories

```
mv folder dest

mv folder/ dest
```

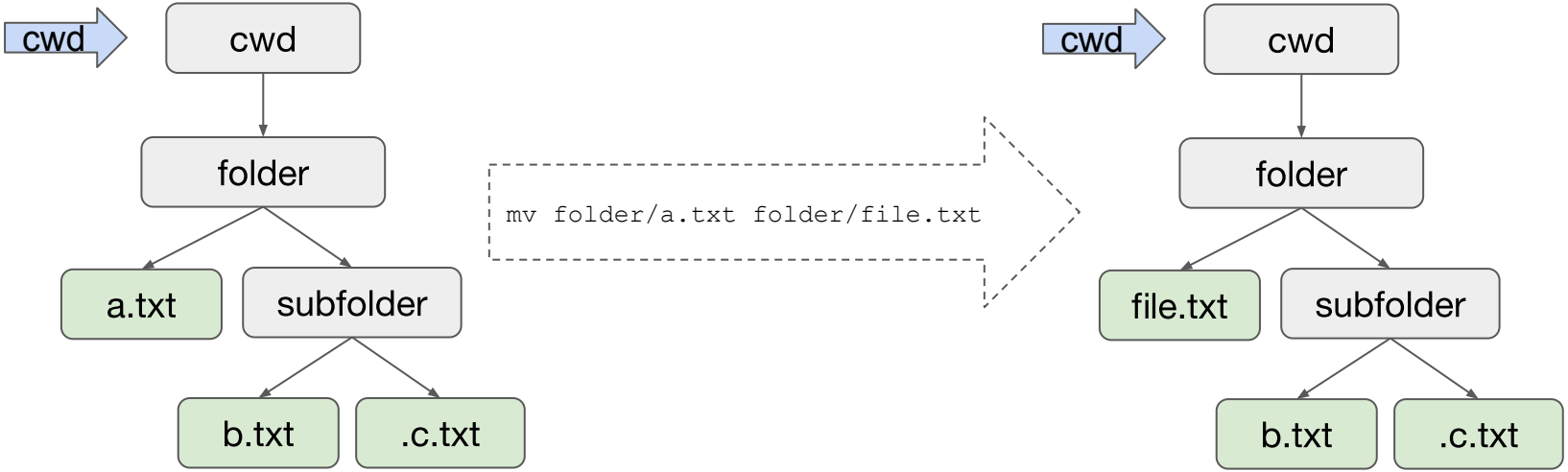


03.04 Renaming files/directories

Common use case is renaming a file.

⇒ if `dest` exists, `mv` will overwrite it!

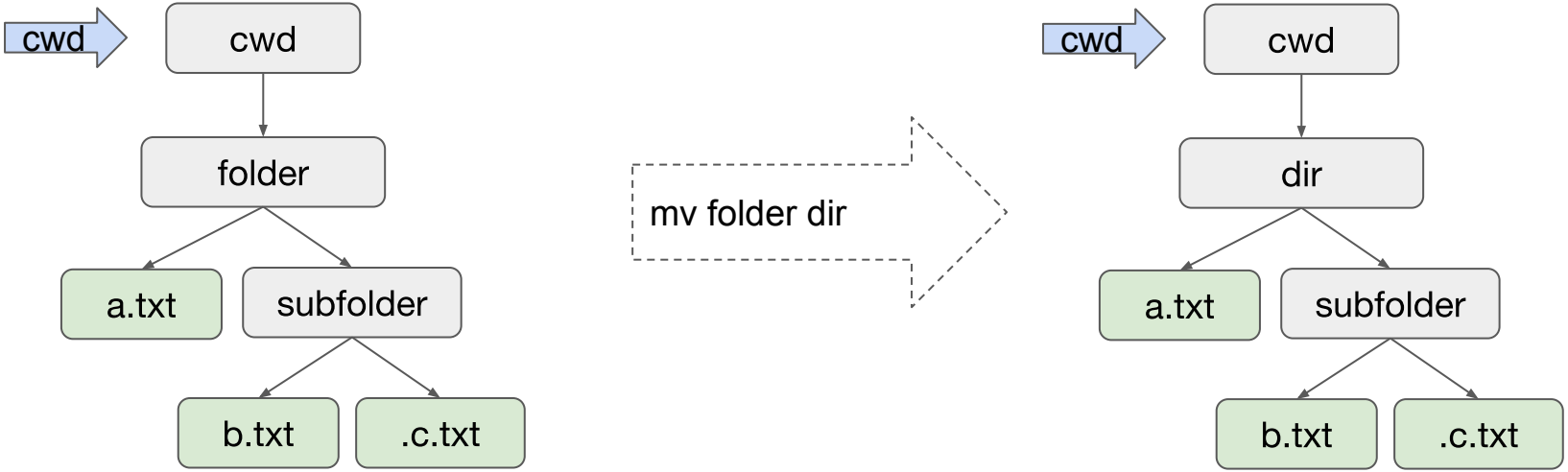
⇒ use `-n` to disable that behavior or `-i` to prompt



03.04 Renaming files/directories

Renaming directories works too if dest does not exist.

⇒ **If dest exists, then mv will put src under dest!**



How to mv or cp a subset of files?

03.05 UNIX wildcards

⇒ match subset of files with placeholders

Examples:

1. copy all JPEG files to images/
2. move all image files to images/
3. list all PDF files in cwd

03.05 UNIX wildcards

`*`	match zero or more arbitrary characters
`?`	match exactly one arbitrary character
`[...]`	match one character within the square brackets
`[a-z]`	match one character of lowercase abc (a, b, c, d, …, z)
`[0-9]`	match one character to a digit 0, 1, …, 9
`[a-zA-Z]`	match lower/upper case alphabet
`[a-zA-Z0-9]`	match one of lower/upper case alphabet or single digit
`[!...]`	negate, i.e. match one character which is not in ...
`[!123]`	match a single character which is not 1,2 or 3

Hint: escape `[, *, ?` using a backslash, e.g. `*` ⇒ `\*`

03.05 UNIX wildcards - example

⇒ matching occurs between file separators

```
/*folder/file?.txt
```

matches:

```
/folder/file1.txt
/subfolder/filea.txt
/subsubfolder/fileB.txt
```

does not match:

```
folder/file.txt
folder/file1.txt
/subfolder/a.txt
/folder/subfolder/file1.txt
```

03.05 UNIX wildcards

Example:

moving all JPEG files to an images/

```
mv *.jpg images/
```

```
mv *?.jpg images/
```

03.05 UNIX wildcards

Example:

moving all JPEG files to an images/

```
mv *.jpg images/
```

would move hidden file .jpg too

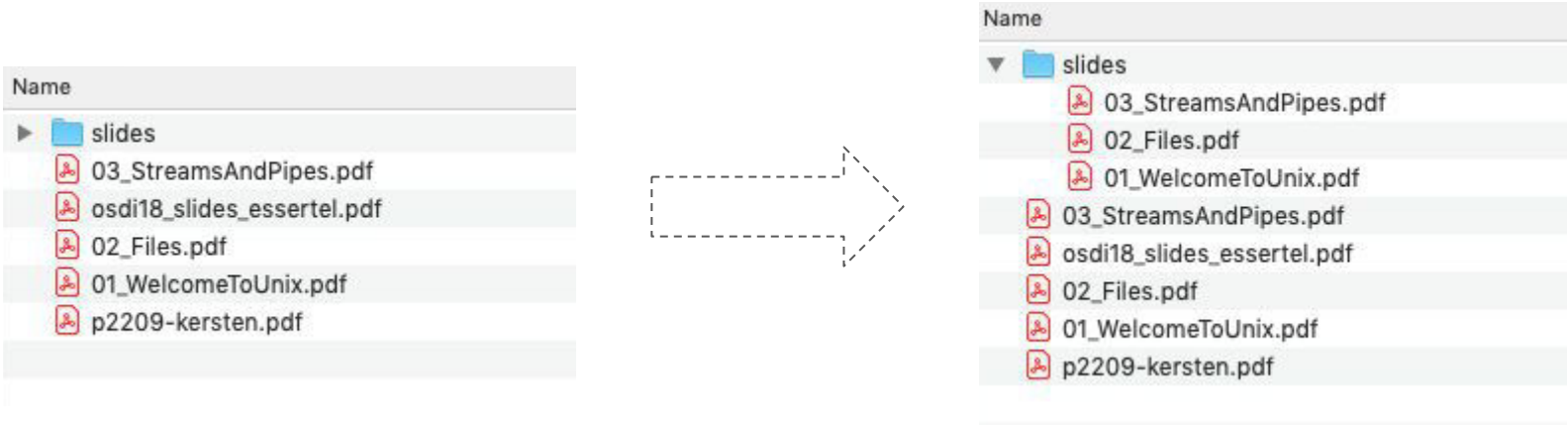
```
mv *?.jpg images/
```

moves only visible JPEG files

03.05 UNIX wildcards

copying all lectures to a folder slides/

```
cp [0-9][0-9]_*.pdf slides/
```



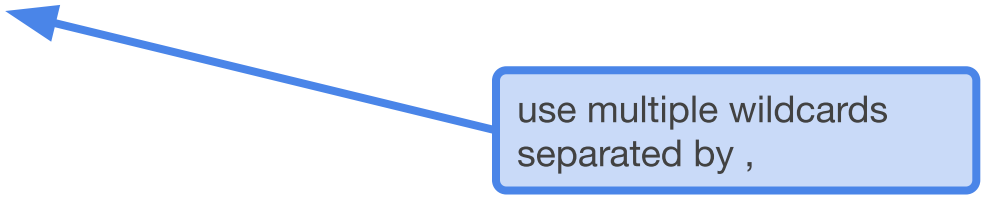
03.05 UNIX wildcards

⇒ what if we want to move both .png and .jpg files?

⇒ curly braces {...} provide a way to execute a command with multiple wildcards

Example:

```
mv {*.png,*.jpg} images/
```



03.05 Brace expansion

`{...}` expands … to multiple arguments

`{*.png,*.jpg}` ⇒ `*.png *.jpg`

Example:

```
mv {*.png,*.jpg} images/
```

⇒ `mv *.png *.jpg images/`

03.05 Brace expansion

Shorter version:

```
mv *.{png,jpg} images/
```

⇒ `mv *.png *.jpg images/`

03.05 Brace expansion

Multiple brace expansions are also possible

Example:

```
ls {lecture,slide*}_??.{pdf,dvi}
```

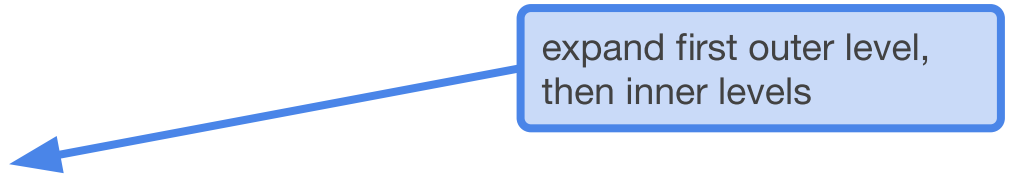
⇒ `ls lecture_??.pdf lecture_??.dvi slide*_??.pdf slide*_??.dvi`

03.05 Brace expansion

Braces can be nested too! ⇒ fast way to generate arguments

Examples:

```
ls {a,b{c,d}}.txt
```




```
⇒ ls a.txt b{c,d}.txt

⇒ ls a.txt bc.txt bd.txt

ls {a,{b,c}}.txt

⇒ ls a.txt b.txt c.txt
```

03.05 Brace expansion

One more example:

Making a backup of a file

```
cp file{,.backup}
```

⇒ `cp file file.backup`

03.05 UNIX wildcards

Limitation:

`cp *.*{,.backup}` does not work! Why?

⇒ `cp *.* *.*.backup`

⇒ Copy command expects a target_file or target_directory. It can not deduce this from the wildcard.

⇒ Always read the man page!

How can we access
the contents of a file?

03.06 File types

files are in the end a collection of 0/1 bits (8 bits = 1 byte).

the file encoding decides what the meaning of the file is

⇒ two categories of encodings: **text** and **binary**

⇒ files serve different purposes:
they represent **data** or executable **code**

03.06 Text files

Print file contents to terminal using terminal encoding

`cat file …`

Popular encodings:

ASCII

UTF-8

03.06 Text files - ASCII

ASCII = American Standard Code for Information Interchange

https://www.asciitable.xyz/

⇒ 7bit encoding of characters

⇒ **one** byte encodes **one** character

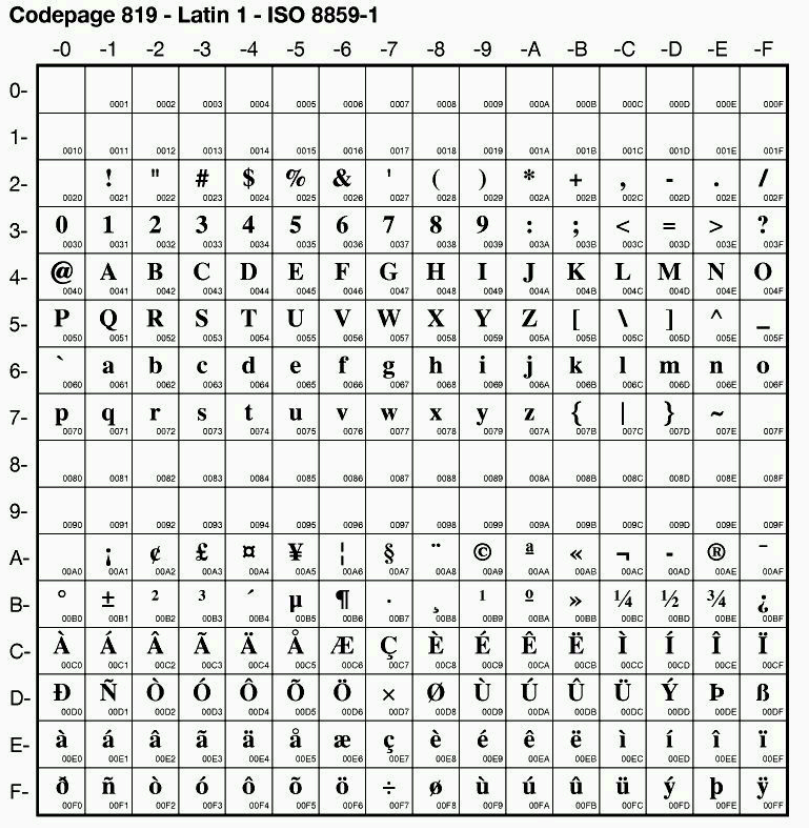
Char	Dec	Oct	Hex	Char	Dec	Oct	Hex	Char	Dec	Oct	Hex
(sp)	32	0040	0x20	@	64	0100	0x40	`	96	0140	0x60
!	33	0041	0x21	A	65	0101	0x41	a	97	0141	0x61
"	34	0042	0x22	B	66	0102	0x42	b	98	0142	0x62
#	35	0043	0x23	C	67	0103	0x43	c	99	0143	0x63
$	36	0044	0x24	D	68	0104	0x44	d	100	0144	0x64
%	37	0045	0x25	E	69	0105	0x45	e	101	0145	0x65
&	38	0046	0x26	F	70	0106	0x46	f	102	0146	0x66
'	39	0047	0x27	G	71	0107	0x47	g	103	0147	0x67
(	40	0050	0x28	H	72	0110	0x48	h	104	0150	0x68
)	41	0051	0x29	I	73	0111	0x49	i	105	0151	0x69
*	42	0052	0x2a	J	74	0112	0x4a	j	106	0152	0x6a
+	43	0053	0x2b	K	75	0113	0x4b	k	107	0153	0x6b
,	44	0054	0x2c	L	76	0114	0x4c	l	108	0154	0x6c
-	45	0055	0x2d	M	77	0115	0x4d	m	109	0155	0x6d
.	46	0056	0x2e	N	78	0116	0x4e	n	110	0156	0x6e
/	47	0057	0x2f	O	79	0117	0x4f	o	111	0157	0x6f
0	48	0060	0x30	P	80	0120	0x50	p	112	0160	0x70
1	49	0061	0x31	Q	81	0121	0x51	q	113	0161	0x71
2	50	0062	0x32	R	82	0122	0x52	r	114	0162	0x72
3	51	0063	0x33	S	83	0123	0x53	s	115	0163	0x73
4	52	0064	0x34	T	84	0124	0x54	t	116	0164	0x74
5	53	0065	0x35	U	85	0125	0x55	u	117	0165	0x75
6	54	0066	0x36	V	86	0126	0x56	v	118	0166	0x76
7	55	0067	0x37	W	87	0127	0x57	w	119	0167	0x77
8	56	0070	0x38	X	88	0130	0x58	x	120	0170	0x78
9	57	0071	0x39	Y	89	0131	0x59	y	121	0171	0x79
:	58	0072	0x3a	Z	90	0132	0x5a	z	122	0172	0x7a
;	59	0073	0x3b	[	91	0133	0x5b	{	123	0173	0x7b
<	60	0074	0x3c	\	92	0134	0x5c	|	124	0174	0x7c
=	61	0075	0x3d	]	93	0135	0x5d	}	125	0175	0x7d
>	62	0076	0x3e	^	94	0136	0x5e	~	126	0176	0x7e
?	63	0077	0x3f	_	95	0137	0x5f				

03.06 Text files - ISO/IEC 8859-1

also known as *Latin alphabet no. 1*

⇒ based on ASCII

⇒ 8 bit encoding, extending ASCII



Codepage 819 - Latin 1 - ISO 8859-1

	-0	-1	-2	-3	-4	-5	-6	-7	-8	-9	-A	-B	-C	-D	-E	-F
0-	0000	0001	0002	0003	0004	0005	0006	0007	0008	0009	000A	000B	000C	000D	000E	000F
1-	0010	0011	0012	0013	0014	0015	0016	0017	0018	0019	001A	001B	001C	001D	001E	001F
2-	0020	! 0021	" 0022	# 0023	$ 0024	% 0025	& 0026	' 0027	(0028	) 0029	* 002A	+ 002B	, 002C	- 002D	. 002E	/ 002F
3-	0 0030	1 0031	2 0032	3 0033	4 0034	5 0035	6 0036	7 0037	8 0038	9 0039	: 003A	; 003B	< 003C	= 003D	> 003E	? 003F
4-	@ 0040	A 0041	B 0042	C 0043	D 0044	E 0045	F 0046	G 0047	H 0048	I 0049	J 004A	K 004B	L 004C	M 004D	N 004E	O 004F
5-	P 0050	Q 0051	R 0052	S 0053	T 0054	U 0055	V 0056	W 0057	X 0058	Y 0059	Z 005A	[005B	\ 005C	] 005D	^ 005E	_ 005F
6-	` 0060	a 0061	b 0062	c 0063	d 0064	e 0065	f 0066	g 0067	h 0068	i 0069	j 006A	k 006B	l 006C	m 006D	n 006E	o 006F
7-	p 0070	q 0071	r 0072	s 0073	t 0074	u 0075	v 0076	w 0077	x 0078	y 0079	z 007A	{ 007B	| 007C	} 007D	~ 007E	007F
8-	0080	0081	0082	0083	0084	0085	0086	0087	0088	0089	008A	008B	008C	008D	008E	008F
9-	0090	0091	0092	0093	0094	0095	0096	0097	0098	0099	009A	009B	009C	009D	009E	009F
A-	00A0	¡ 00A1	¢ 00A2	£ 00A3	¤ 00A4	¥ 00A5	¦ 00A6	§ 00A7	¨ 00A8	© 00A9	ª 00AA	« 00AB	¬ 00AC	- 00AD	® 00AE	¯ 00AF
B-	° 00B0	± 00B1	² 00B2	³ 00B3	´ 00B4	µ 00B5	¶ 00B6	· 00B7	¸ 00B8	¹ 00B9	º 00BA	» 00BB	¼ 00BC	½ 00BD	¾ 00BE	¿ 00BF
C-	À 00C0	Á 00C1	Â 00C2	Ã 00C3	Ä 00C4	Å 00C5	Æ 00C6	Ç 00C7	È 00C8	É 00C9	Ê 00CA	Ë 00CB	Ì 00CC	Í 00CD	Î 00CE	Ï 00CF
D-	Ð 00D0	Ñ 00D1	Ò 00D2	Ó 00D3	Ô 00D4	Õ 00D5	Ö 00D6	× 00D7	Ø 00D8	Ù 00D9	Ú 00DA	Û 00DB	Ü 00DC	Ý 00DD	Þ 00DE	ß 00DF
E-	à 00E0	á 00E1	â 00E2	ã 00E3	ä 00E4	å 00E5	æ 00E6	ç 00E7	è 00E8	é 00E9	ê 00EA	ë 00EB	ì 00EC	í 00ED	î 00EE	ï 00EF
F-	ð 00F0	ñ 00F1	ò 00F2	ó 00F3	ô 00F4	õ 00F5	ö 00F6	÷ 00F7	ø 00F8	ù 00F9	ú 00FA	û 00FB	ü 00FC	ý 00FD	þ 00FE	ÿ 00FF

03.06 Text files - ASCII and UTF-8 encoding

UTF = Unicode Transformation Format

8 refers to 8bit as encoding element

UTF-8: ASCII backward compatible encoding with variable length code

⇒ UTF-8 is the **defacto** standard encoding today

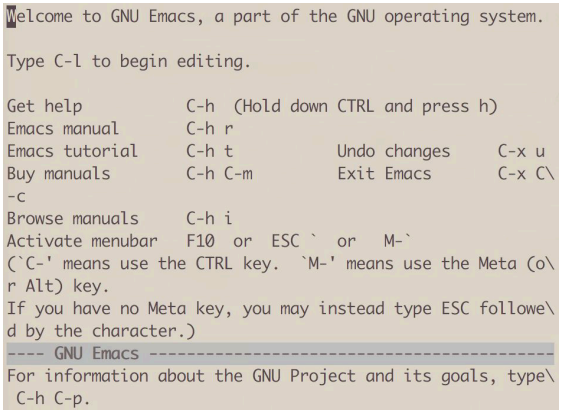
03.06 Text files - UTF-8 encoding

number of bytes		Byte 1	Byte 2	Byte 3	Byte 4
1	ASCII chars	0xxxxxxx			
2	Latin, Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Thaana, ...	110xxxxx	10xxxxxx		
3	Chinese, Japanese, Korean, ….	1110xxxx	10xxxxxx	10xxxxxx	
4	Math symbols, Emojis	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

To represent a character in UTF-8 1-4 bytes are required!

03.06 Editing text files

vim



emacs

nano

⇒ Lab01: Learning how to use VIM

03.07 Binary files

Can be either data files or executable machine code

⇒ in Unix programs (executables) don't have a file ending typically

`hexdump file` allows to print a binary file in hexadecimal format

⇒ output is called a *hexdump*

03.07 hexdump

first column is byte counter in hex

```
hexdump penguin.jpg

0000000 ff d8 ff e1 12 d4 45 78 69 66 00 00 4d 4d 00 2a
0000010 00 00 00 08 00 0c 01 00 00 03 00 00 00 01 08 00
0000020 00 00 01 01 00 03 00 00 00 01 0c 00 00 00 01 02
0000030 00 03 00 00 00 03 00 00 00 9e 01 06 00 03 00 00
0000040 00 01 00 02 00 00 01 12 00 03 00 00 00 01 00 01
0000050 00 00 01 15 00 03 00 00 00 01 00 03 00 00 01 1a
0000060 00 05 00 00 00 01 00 00 00 a4 01 1b 00 05 00 00
0000070 00 01 00 00 00 ac 01 28 00 03 00 00 00 01 00 02
0000080 00 00 01 31 00 02 00 00 00 24 00 00 00 b4 01 32
0000090 00 02 00 00 00 14 00 00 00 d8 87 69 00 04 00 00
...
```

03.07 hexdump vs. cat

cat file.txt

```
Tux loves CS6
```

hexdump -c file.txt

```
0000000   T   u   x       l   o   v   e   s       C   S   6  \n
000000e
```

hexdump file.txt

```
0000000 54 75 78 20 6c 6f 76 65 73 20 43 53 36 0a
000000e
```

-c lets hexdump print bytes in terminal encoding

hexdump has several more interesting options
⇒ read the man page

03.08 file

`file` `file` prints (if able) a short info what type of file file is

⇒ helpful to determine file type because the ending does not correspond to the file type in general

⇒ e.g. image.png could be in fact an executable

Example:

```
file penguin.jpg
```

```
penguin.jpg: JPEG image data,
Exif standard: [TIFF image
data, big-endian,
direntries=12, height=3072,
bps=0,
PhotometricIntepretation=RGB,
orientation=upper-left,
width=2048], progressive,
precision 8, 512x513, frames 3
```

03.09 Unix vocabulary so far

```
cd
ls
touch
rm
mkdir
rmdir
```

```
mv
cp
cat
hexdump
file
```

04 Users and Permissions

CS6 Practical System Skills

Fall 2019

Leonhard Spiegelberg *lspiegel@cs.brown.edu*

04.01 Permissions

UNIX is a multi-user system.

How do you protect files from other users, the world?

How do you share files with other users?

How do you protect one from oneself?

04.01 Permissions

Each file in Unix has 3 permissions:

read	the file can be read, i.e. its contents displayed
write	the file can be modified or deleted
execute	the file can be run (i.e. executables or scripts)

04.01 Permissions for directories

Because directories are also files, they have read, write, or execute permissions too. The meaning differs though:

permission	file	directory
read	Allows file to be read.	Allows file names in the directory to be read.
write	Allows file to be modified.	Allows entries to be modified within the directory.
execute	Allows file to be executed.	Allows access to contents and metadata for entries in the directory.

04.01 Users and permissions

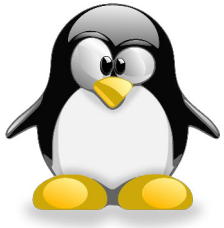
Each file is owned by a user
⇒ typically the creator

In addition, each file belongs to a group ⇒ smallest group: the user

Three categories of users can have permissions to a file

1.) Owner
2.) Group
3.) Other

04.01 Users and permissions



owner
creator of the file

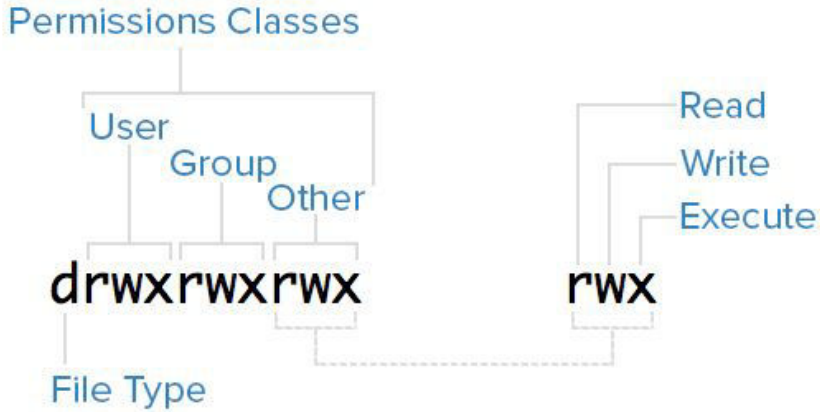
group
multiple users

other
public, world

⇒ UNIX allows you to set (for each file) separate read/write/execute permissions for each party

04.02 ls longformat

owner and user are usually the same!
Terms are used interchangeably here often.

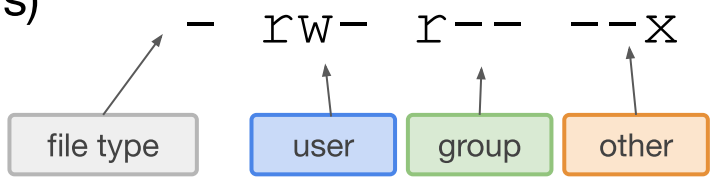



```
ls -l

total 88

-rw-r--r-- 1 sealion  friends        14  9 Sep 8:01 file.txt

-rw-r--r-- 1 sealion  friends  40390  9 Sep 9:00 penguin.jpg
```

permission string | owner | group

04.02 Permissions

permission string (10 characters)



filetype	symbol
regular file	-
directory	d
symbolic link	l
pipe	p
socket	s
block device	b
char device	c

permission	symbol
read	r
write	w
e**x**ecute	x

04.03 Setting permissions - chmod

`chmod mode file …`

change mode, i.e. set or update file permissions
⇒ only the owner (or root) can run this command for a file

⇒ mode can be either a number (numeric mode) or a combination of symbols

04.04 chmod - symbolic mode

party	symbol
user	u
group	g
other	o
all	a

action	symbol
add permission	+
remove permission	-
set to	=

permission	symbol
read	r
write	w
e**x**ecute	x

Example:

```
chmod u=rw,g=rx,o= file.txt
```

04.05 chmod - numeric mode

Instead of using symbols, chmod can be used with an even short syntax using the following encoding.

Octal	Binary	String	Description
0	000	---	no permissions
1	001	--x	execute only
2	010	-w-	write only
3	011	-wx	write and execute
4	100	r--	read only
5	101	r-x	read and execute
6	110	rw-	read and write
7	111	rwx	read, write and execute

```
chmod u=rw,g=rx,o= file.txt ⇒ chmod 650 file.txt
```

More details next lecture!

Preview next lecture

⇒ Commonly used permissions

⇒ Practical use cases involving permissions

⇒ Special Linux permissions

⇒ Streams & Pipes

Preview HW01:

Out today!

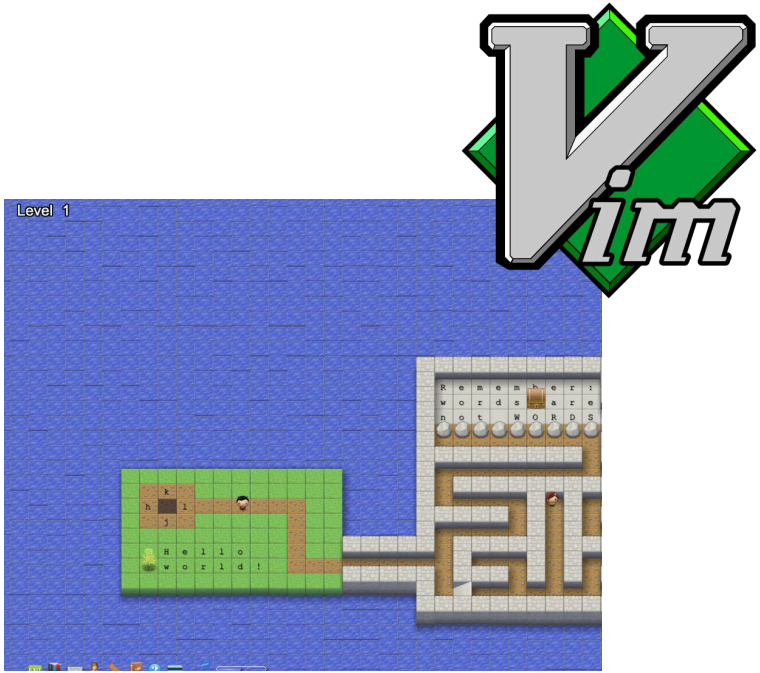
Due: next Tue 17th September, 4pm on Gradescope

Topic: Files, permissions, VIM

Preview Lab01:

Today: 8pm -10pm @ CIT 201

Topic: Installation / Setup / VIM



⇒ If you don't know VIM,
please attend the lab!

End of lecture.

Next class: Thu, 4pm-5:20pm @ CIT 477