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 ~
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?

```bash
ls Documents

ls

ls -a
```
## 03.00 Recap quiz

<table>
<thead>
<tr>
<th>ls Documents</th>
</tr>
</thead>
<tbody>
<tr>
<td>ls: Documents: No such file or directory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ls</th>
</tr>
</thead>
<tbody>
<tr>
<td>mail</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ls -a</th>
</tr>
</thead>
<tbody>
<tr>
<td>. .. mail .backups</td>
</tr>
</tbody>
</table>
CS6 Practical System Skills
Fall 2019
Leonhard Spiegelberg lspiegel@cs.brown.edu
# 03.01 Creating and deleting files

<table>
<thead>
<tr>
<th>files</th>
<th>directories</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>touch file</strong></td>
<td>create a new file if it doesn’t exist, update access/modification time</td>
</tr>
<tr>
<td><strong>mkdir directory</strong></td>
<td>create a new directory, fails if directory already exists</td>
</tr>
<tr>
<td><strong>rm file</strong></td>
<td>remove file</td>
</tr>
<tr>
<td><strong>rmdir directory</strong></td>
<td>remove directory, if it is empty</td>
</tr>
</tbody>
</table>
03.02 Working with directories

create recursive directories on the fly

-\texttt{p} : create intermediate directories as required

Example:

\texttt{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

\texttt{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/
```

Type `y` or `n`, then press ENTER to decide in the prompt whether to perform action or not.
03.03 Deleting files in a tree

delete directories, including all files and subdirectories in them

-\texttt{r} \quad \text{recursively remove files in hierarchy}

-\texttt{f} \quad \text{force removal, silent prompts & warnings}

Example:

\texttt{rm -rf folder/}

This is the danger zone, run this only when you KNOW what happens.
Back to the terminal...
## 03.04 Organizing files

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mv src ... dest</code></td>
<td>move file from src location to dest location ⇒ can be used to rename a file</td>
</tr>
<tr>
<td><code>cp src ... dest</code></td>
<td>copy file from src location to dest location</td>
</tr>
</tbody>
</table>

- `mv` command works for directories too.
- `cp` command 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/ .

.cwd

folder

a.txt  subfolder

b.txt  .c.txt

.cwd

folder

a.txt  subfolder

b.txt  .c.txt

.cwd

folder

a.txt

subfolder

b.txt  .c.txt

.cwd

folder

a.txt

subfolder

b.txt

.c.txt

. 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

same effect
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

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

**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/

```bash
mv *.jpg images/
mv *?.jpg images/
```
03.05 UNIX wildcards

Example:
moving all JPEG files to an images/

mv *.jpg images/

mv *?.jpg images/

would move hidden file .jpg too

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

\{ \ldots \} expands \ldots to multiple arguments

\{ *.png, *.jpg \} \Rightarrow *.png *.jpg

Example:

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

⇒ mv *.png *.jpg images/
Shorter version:

```bash
mv *.png *.jpg images/
```

⇒ `mv *.png *.jpg images/`
03.05 Brace expansion

Multiple brace expansions are also possible

Example:

```bash
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?
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
also known as *Latin alphabet no. 1*

⇒ based on ASCII

⇒ 8 bit encoding, extending ASCII
UTF = Unicode Transformation Format

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

<table>
<thead>
<tr>
<th>number of bytes</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ASCII chars</td>
<td>0xxxxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Latin, Greek, Cyrillic, Coptic, Armenian, Hebrew, Arabic, Syriac, Thaana, ...</td>
<td>110xxxxx</td>
<td>10xxxxxx</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Chinese, Japanese, Korean, ...</td>
<td>1110xxxxx</td>
<td>10xxxxxx</td>
<td>10xxxxxx</td>
</tr>
<tr>
<td>4</td>
<td>Math symbols, Emojis</td>
<td>11110xxx</td>
<td>10xxxxxx</td>
<td>10xxxxxx</td>
</tr>
</tbody>
</table>

To represent a character in UTF-8 1-4 bytes are required!
03.06 Editing text files

⇒ 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

hexdump penguin.jpg

```
00000000 ff d8 ff e1 12 d4 45 78 69 66 00 00 4d 4d 00 2a
00000010 00 00 00 08 00 0c 01 00 00 03 00 00 00 01 08 00
00000020 00 00 01 01 00 03 00 00 00 01 0c 00 00 00 01 02
00000030 00 03 00 00 00 03 00 00 00 9e 01 06 00 03 00 00
00000040 00 01 00 02 00 00 01 12 00 03 00 00 00 01 00 01
00000050 00 00 01 15 00 03 00 00 00 01 00 03 00 00 01 1a
00000060 00 05 00 00 00 01 00 00 00 a4 01 1b 00 05 00 00
00000070 00 01 00 00 00 ac 01 28 00 03 00 00 00 01 00 02
00000080 00 00 01 31 00 02 00 00 00 24 00 00 00 b4 01 32
00000090 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** |
| 00000000 T u x l o v e s C S 6 \n 0000000e |

| **hexdump file.txt** |
| 00000000 54 75 78 20 6c 6f 76 65 73 20 43 53 36 0a 0000000e |

- `-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, PhotometricInterpretation=RGB, orientation=upper-left, width=2048], progressive, precision 8, 512x513, frames 3
03.09 Unix vocabulary so far

<table>
<thead>
<tr>
<th>Command</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>cd</td>
<td>mv</td>
</tr>
<tr>
<td>ls</td>
<td>cp</td>
</tr>
<tr>
<td>touch</td>
<td>cat</td>
</tr>
<tr>
<td>rm</td>
<td>hexdump</td>
</tr>
<tr>
<td>mkdir</td>
<td>file</td>
</tr>
<tr>
<td>rmdir</td>
<td></td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>permission</th>
<th>file</th>
<th>directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>Allows file to be read.</td>
<td>Allows file names in the directory to be read.</td>
</tr>
<tr>
<td>write</td>
<td>Allows file to be modified.</td>
<td>Allows entries to be modified within the directory.</td>
</tr>
<tr>
<td>execute</td>
<td>Allows file to be executed.</td>
<td>Allows access to contents and metadata for entries in the directory.</td>
</tr>
</tbody>
</table>
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

- **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

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

permission string (10 characters)

<table>
<thead>
<tr>
<th>filetype</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular file</td>
<td>-</td>
</tr>
<tr>
<td>directory</td>
<td>d</td>
</tr>
<tr>
<td>symbolic link</td>
<td>l</td>
</tr>
<tr>
<td>pipe</td>
<td>p</td>
</tr>
<tr>
<td>socket</td>
<td>s</td>
</tr>
<tr>
<td>block device</td>
<td>b</td>
</tr>
<tr>
<td>char device</td>
<td>c</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>permission</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>r</td>
</tr>
<tr>
<td>write</td>
<td>w</td>
</tr>
<tr>
<td>execute</td>
<td>x</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>party</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>u</td>
</tr>
<tr>
<td>group</td>
<td>g</td>
</tr>
<tr>
<td>other</td>
<td>o</td>
</tr>
<tr>
<td>all</td>
<td>a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>action</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>add permission</td>
<td>+</td>
</tr>
<tr>
<td>remove permission</td>
<td>-</td>
</tr>
<tr>
<td>set to</td>
<td>=</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>permission</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>r</td>
</tr>
<tr>
<td>write</td>
<td>w</td>
</tr>
<tr>
<td>execute</td>
<td>x</td>
</tr>
</tbody>
</table>

Example:

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

defines the file permissions as read, write, and execute for user, group, and other respectively.

sets mask to -rw-r-x---
### 04.05 chmod - numeric mode

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

<table>
<thead>
<tr>
<th>Octal</th>
<th>Binary</th>
<th>String</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>000</td>
<td>---</td>
<td>no permissions</td>
</tr>
<tr>
<td>1</td>
<td>001</td>
<td>--x</td>
<td>execute only</td>
</tr>
<tr>
<td>2</td>
<td>010</td>
<td>-w-</td>
<td>write only</td>
</tr>
<tr>
<td>3</td>
<td>011</td>
<td>-wx</td>
<td>write and execute</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>r--</td>
<td>read only</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>r-x</td>
<td>read and execute</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
<td>rw-</td>
<td>read and write</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>rwx</td>
<td>read, write and execute</td>
</tr>
</tbody>
</table>

```bash
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 Lab 01:

**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