CSCI 2132
Software Development

Lecture 4:

Files and Directories

Instructor: Vlado Keselj

Faculty of Computer Science

Dalhousie University

Previous Lecture

- Some hardware concepts
- Main UNIX concepts, Shells
- Logging in, PuTTY
- Some basic utilities and commands
  - date, clear, passwd, man
- Shell metacharacters
- ‘cat’ example, file redirection
- Logging out

Files and Directories

- Many concepts in Unix are either a **file** or a **process**
- **File** is a stream of bytes
- Many devices and constructs are seen as files:
  - regular files, stdin, stdout, stderr, keyboard, monitor, hard disk, CD/DVD, . . .
- File is a good example of **abstraction**
- File is described by a general **interface**

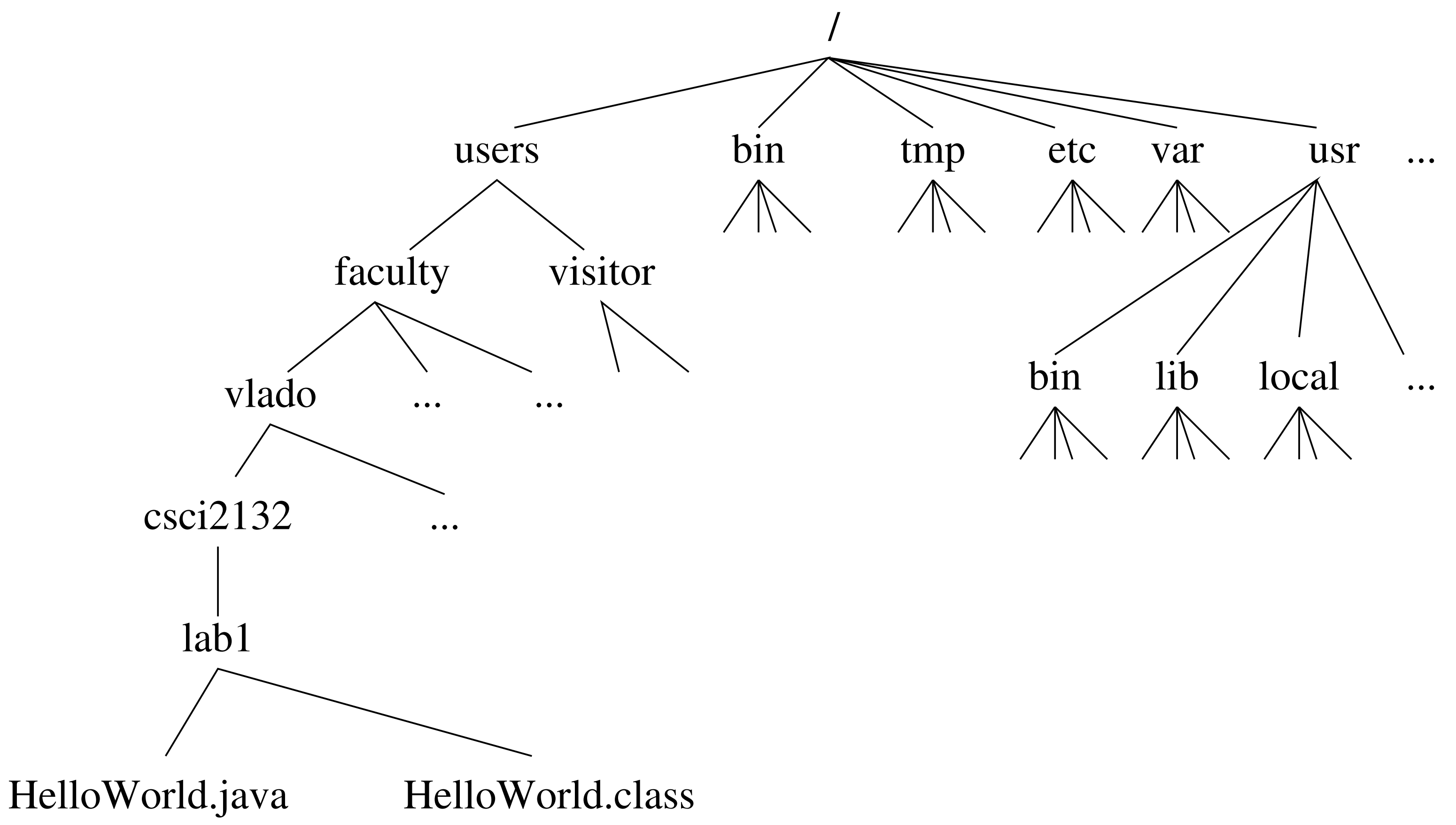
Seven Types of Files

1. Regular files
2. Directory files
3. Buffered special files (block devices)
4. Unbuffered special files (character devices)
5. Symbolic links
6. Pipes (named pipes)
7. Sockets

- `ls -l` command reveals a file type: -, d, b, c, l, p, s
- Example:

```
drwxr-xr-x 2 vlado csfac 4096 Sep 13 06:24 c
-rw-r--r-- 1 vlado csfac    0 Sep 13 06:34 file
```

Navigating Directory Structure



Some Notions in Directory Structure

- A tree with **root directory** (`/`)
- If a directory A contains directly directory B:
  - A is **parent directory** of B
  - B is **subdirectory** of A
- Each directory has two special directory entries:
  - dot (`.`) — the directory itself
  - dot-dot (`..`) — the parent directory

Pathname (Path)

- Each file has a **name**
- Files can have the same name if they are in different directories
  - Example: see `bin` in the previous figure
- To distinguish files with the same name, we use pathnames
- **Pathname** (or **path**) is a sequence of directories, finishing with a file name
- Directories are separated using character slash (`/`)
- Example:

```
/users/faculty/vlado/csci2132/lab1/HelloWorld.java
```

Two Kinds of Paths

- **Absolute path** starts from root (initial slash `/`), examples:

```
/usr/bin
/users/faculty/vlado/csci2132/lab1/HelloWorld.java
```

- **Relative path** starts from the **current directory**; examples (if the current directory is ‘vlado’):

```
csci2132
csci2132/lab1/HelloWorld.java
./csci2132/lab1/HelloWorld.java
../../visitor
./a.out
```

Parts of Pathname

- Pathname: dirname and basename
- Example commands:

```
$ basename /home/ed/file.txt
file.txt
$ basename /home/ed/file.txt .txt
file
$ dirname /home/ed/file.txt
/home/ed
```

- Note: blue text above is system output and red text is our input; we will use `$` or `>` as shell prompt.

Useful Commands related to Directories

- `ls` *`paths`* — list directory contents
- `pwd` — print working directory
- `cd` *`path`* — change directory
- `mkdir` *`dirs`* — make directory(ies)
- `mkdir -p` *`paths`* — whole paths, no errors
- `rmdir` *`dirs`* — remove empty directory(ies)
- `mv` *`path1 path2`* — move or rename directory or file
- `mv -i` *`path1 path2`* — prompt before overwrite
- `rm` *`paths`* — remove files but can remove directories with option `-r`; useful to consider `-f` and `-i`
- `tree` *`paths`* — note: not a standard Unix command

A Small Exercise

Let us consider the following commands:

```
$ pwd
/home/ed
$ mkdir tmp
$ cd tmp
$ mkdir a b c
$ mkdir -p a/a1 a/a2/a21 a/a2/a22
$ cd a/a2/a22
```

What is our absolute current directory?

What directory is `..`?

Do the following directories exist and what are their absolute paths: `..`, `../../b`, and `../../../c` ?

File Manipulation

- `cat` *files* — showing textual file(s) content
- `more` *files* — showing textual file content, paged
- `head` *files* — showing textual file content, first part
- `tail` *files* — showing textual file content, last part
- `vi`, `emacs`, `pico`, `nano` — file editors
- `wc` *files* — word count
  - learn about `-c`, `-w`, and `-l` options of `wc`

File Permissions

- We will discuss the concepts of:
  - users and groups
  - different types of permissions on files

Users, Usernames and UserIDs

- Used to protect files and processes between different users
- Every user has a unique **username**, which is a text string
- Try command: `whoami`
- The system uses numeric userid, which we will call just **userID** (username is for string id)
- Try command: `id -u`

Groups

- Every UNIX user is a member of a group
- A user can be member of multiple groups, but one is effective for a process
- Each group has a unique groupname and groupID
- Command to list groups user is member of: `groups`
- Command for more complete information: `id`
- Each process, including shell, has one effective userID and groupID
- Each file is owned by one user and one group: **file owner** and **file group**

File Permissions

- Each file has 3 sets of permissions:
  - file owner permissions (u)
  - file group permissions, (but not the user), (g)
  - permissions for others, (not user and not the group) (o)
- For each set, there are three true/false permissions:
  - read (r)
  - write (w)
  - execute (x)
- What these permissions mean for regular files and directories?