1XC3 - Winter 2026
Computer Science Practice and Experience: Development Basics
Lecture Notes

Yingying Wang

Shell & Command Lines

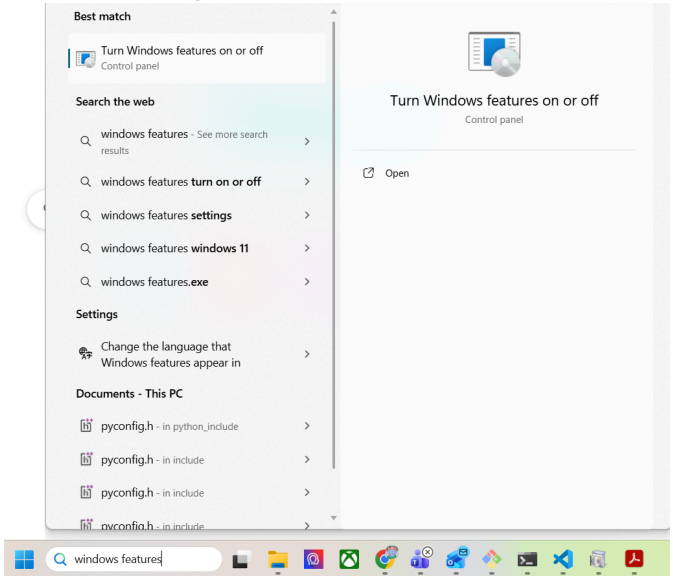
1. Introduction
1.1 What is Shell?

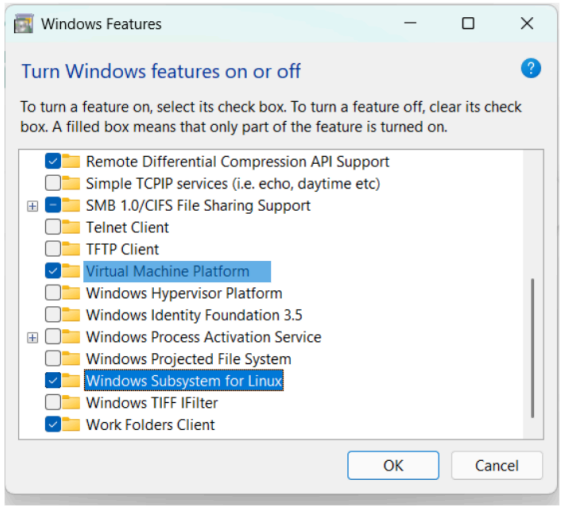
- A program takes keyboard commands and passes them to the operating system to carry out.
- It provides Command Line Interface CLI instead of interactive Graphical User Interface GUI

1.2 How to practice shell commands?

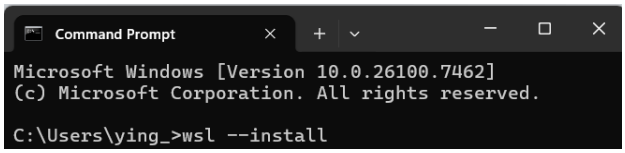
- If you have a Windows machine
  - Install Linux
  - WSL: Windows Subsystem for Linux (recommended)
  - Cygwin
  - Msys2
  - MinGW
  - Git Bash
- If you have a Mac or Linux machine
  - Systems come with a shell that you can practice in

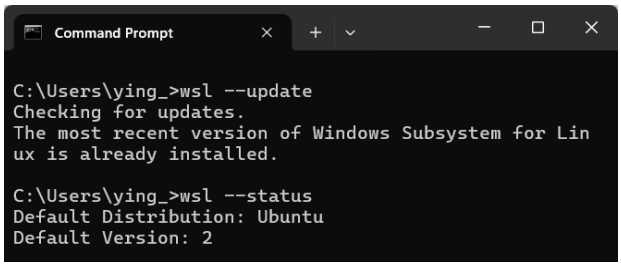
1.3 Installing WSL (for windows users)



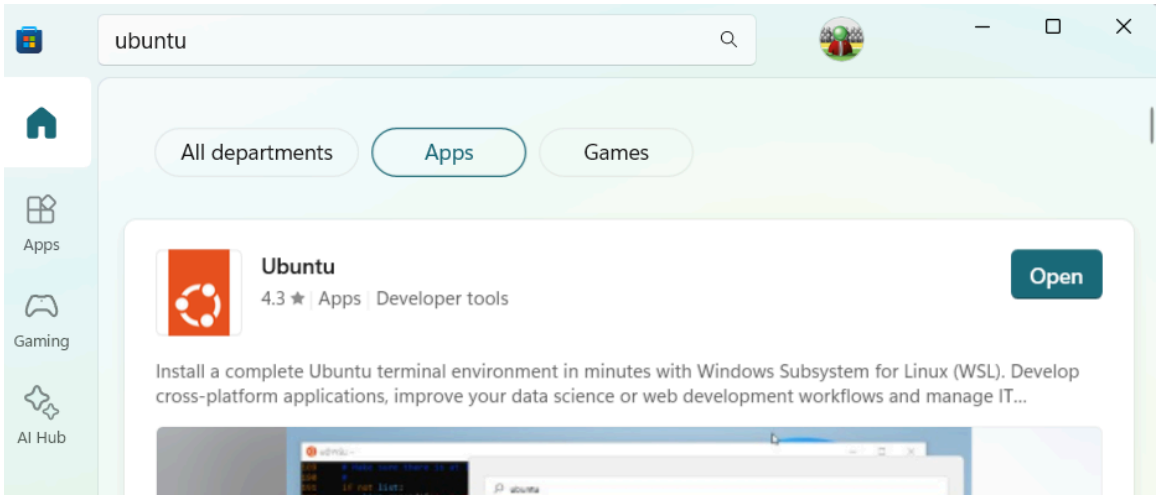


- Search “Windows Features”, open the dialog, and turn on:
  “Virtual Machine Platform”
  “Windows Subsystem for Linux”
- Search “Command Prompt”, open the command prompt window, and install WSL by typing in command:
  `wsl --install`
  other commands you can choose to use:
  `wsl --update`
  `wsl --status`





- Install Linux in Windows Store
  search “Ubuntu” in the Windows Store and click install.



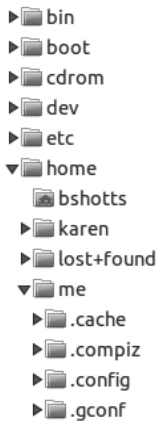
- Use Linux on your Windows
  Click “Open” once Ubuntu is installed in Windows Store
  or type in “WSL” in search bar, open WSL
  Set up your username and password

1.4 Try some simple commands

```
date
cal
free
history
exit
```

1

2. Navigation/Exploring the System



2.1 Path

Specify path names in two different ways

- absolute
  / indicates root, e.g. `/home/karen`
- relative
  . indicates current working directory
  .. indicates parent of the current working directory

Some Useful shortcuts

- ~ indicates user's home directory
- `~username` indicates home directory of `username`
- - indicates previous working directory

2.2 Change Directory `cd dir`

Note that `dir` could be an absolute one or a relative one, or a shortcut
Try the following commands:

`cd /usr/bin`	change current working directory to `/usr/bin`
`cd ..`	go up to `/usr/bin` 's parent, i.e. `/usr`
`cd -`	change working directory to the previous one, i.e. `/usr/bin`
`cd ~`	change current working directory to user home

2

Now create a new folder `1xc3` using `mkdir`, and continue the navigation, replacing `username` with your own user name.

```
mkdir 1xc3
cd ./1xc3
cd
cd 1xc3
cd /
cd ~username
```

3

Question: in the examples above, where does `cd` go?

2.3 Print name of the Current Directory `pwd`

Try the following commands at different locations:

```
pwd
```

4

2.4 List Directory Contents `ls -options arguments`

Option	Long option	Description
-a	--all	List all files, even those with names that begin with a period, which are normally not listed (that is, hidden).
-A	--almost-all	Like the -a option except it does not list . (current directory) and .. (parent directory).
-d	--directory	Ordinarily, if a directory is specified, ls will list the contents of the directory, not the directory itself. Use this option in conjunction with the -l option to see details about the directory rather than its contents.
-F	--classify	This option will append an indicator character to the end of each listed name. For example, it will append a forward slash (/) if the name is a directory.
-h	--human-readable	In long format listings, display file sizes in human-readable format rather than in bytes.
-l		Display results in long format.
-r	--reverse	Display the results in reverse order. Normally, ls displays its results in ascending alphabetical order.
-S		Sort results by file size.
-t		Sort by modification time.

Try the following commands. What are they doing and what are the outputs?

```
ls
ls /usr
ls ~/1xc3  /usr
ls -l
ls -lt
ls -lt  --reverse
ls -a
```

5

To understand the output of `ls`, for example

```
-rw-r--r-- 1 caaml caaml 19773 Jan  6 16:02 test.txt
```

It means:

`-rw-r--r--`	File type & permission: The first character – indicates regular file, if it is `d`, it means directory, or if it is `l`, it indicates symbolic link; `rw-` indicates read/write access for the file owner; `r--` indicates read permission for members of the file group; `r--` indicates read permission for everyone else.
`1`	Number of hard links for a file, or Number of subdirectories for a directory
`caaml`	Username owns the file
`caaml`	Group name owns the file
`19773`	File size in bytes
`Jan  6 16:02`	Last modification time
`test.txt`	File name

2.5 Determine File Type `file filename`

Try the following commands, examining the output of a picture and a text file. What are their outputs?

```
file logo.png
file test.txt
```
6

2.6 View File Content by Page `less filename`

Try the following command:

```
less test.txt
```
7

It displays the content in `test.txt` page by page. Use the following keys to navigate pages:

PAGE UP or b	Scroll back one page
PAGE DOWN or space	Scroll forward one page
Up arrow	Scroll up one line
Down arrow	Scroll down one line
G	Move to the end of the text file
1G or g	Move to the beginning of the text file
/characters	Search forward to the next occurrence of characters
n	Search for the next occurrence of the previous search
h	Display help screen
q	Quit less

3. Manipulating Files and Directories

3.1 Wildcards / Globbing

Wildcards are special characters to help you rapidly specify multiple filenames based on patterns of characters. Using wildcards is also known as globbing.

Wildcard	Meaning
*	Matches any characters
?	Matches any single character
[*characters*]	Matches any character that is a member of the set *characters*
[!*characters*]	Matches any character that is not a member of the set *characters*
[[:*class*:]]	Matches any character that is a member of the specified *class*

Character class	Meaning
[:alnum:]	Matches any alphanumeric character
[:alpha:]	Matches any alphabetic character
[:digit:]	Matches any numeral
[:lower:]	Matches any lowercase letter
[:upper:]	Matches any uppercase letter

What do these mean?

```
*
g*
b*.txt

Data???

[abc]*
BACKUP.[0-9][0-9][0-9]

[[:upper:]]*
[![:digit:]]*
*[[:lower:]123]
```

3.2 Create Directories *mkdir dir...*

```
cd 1xc3
mkdir myExam
mkdir mySlide myBook myNote
ls
ls my*
ls *e
```

3.3 Copy Files and Directories *cp*

- Copy a single file or directory *item1* to *item2*: *cp item1 item2*
- Copy multiple items to directory *dir*: *cp item... dir*

Command	Results
cp *file1 file2*	Copy *file1* to *file2*. If *file2* exists, it is overwritten with the contents of *file1*. If *file2* does not exist, it is created.
cp -i *file1 file2*	Same as previous command, except that if *file2* exists, the user is prompted before it is overwritten.
cp *file1 file2 dir1*	Copy *file1* and *file2* into directory *dir1*. The directory *dir1* must already exist.
cp *dir1/* dir2*	Using a wildcard, copy all the files in *dir1* into *dir2*. The directory *dir2* must already exist.
cp -r *dir1 dir2*	Copy the contents of directory *dir1* to directory *dir2*. If directory *dir2* does not exist, it is created and, after the copy, will contain the same contents as directory *dir1*. If directory *dir2* does exist, then directory *dir1* (and its contents) will be copied into *dir2*.

Try the following commands, `logo.png` is copied twice to `myNote` folder with different names.

```
cp ../logo.png myNote/logoNote.png
cp ../logo.png myNote
ls myNote
```
9

3.4 Move / Rename Files and Directories `mv`

- Move or rename a file or directory `item1` to `item2`: `mv item1 item2`
- Move multiple items to directory `dir`: `mv item... dir`

Command	Results
mv *file1 file2*	Move *file1* to *file2*. **If *file2* exists, it is overwritten with the contents of *file1***. If *file2* does not exist, it is created. **In either case, *file1* ceases to exist**.
mv -i *file1 file2*	Same as the previous command, except that if *file2* exists, the user is prompted before it is overwritten.
mv *file1 file2 dir1*	Move *file1* and *file2* into directory *dir1*. The directory *dir1* must already exist.
mv *dir1 dir2*	If directory *dir2* does not exist, create directory *dir2* and move the contents of directory *dir1* into *dir2* and delete directory *dir1*. If directory *dir2* does exist, move directory *dir1* (and its contents) into directory *dir2*.

Try the following commands:

```
mv myNote/logo.png myNote/logoRename.png
ls myNote
mv myNote/logoRename.png  mySlide
ls myNote mySlide
```
10

The first `mv` command renames the png file, while the second `mv` command moves it from `myNote` to `mySlide`.

3.5 Remove Files and Directories `rm item...`

Command	Results
`rm file1`	Delete *file1* silently.
`rm -i file1`	Same as the previous command, except that the user is prompted for confirmation before the deletion is performed.
`rm -r file1 dir1`	Delete *file1* and *dir1* and its contents.
`rm -rf file1 dir1`	Same as the previous command, except that if either *file1* or *dir1* does not exist, `rm` will continue silently.

Options: `-i` for interactive prompt, `-r` for recursive deleting embedded directories, `-f` indicates by force, `-v` means verbose, displaying deletion information.
ATTENTION: `rm` command could be risky to play around. Make sure you don't accidentally remove anything important in your system.
Understand what will be removed by the following commands:

```
rm mySlide/logoRename.png
rm -r myNote  mySlide
ls
```
11

3.6 Create Links `ln`

3.6.1 Two types of links

- Hard Links
  - every file has a single hard link that gives the file its name
  - indistinguishable from the file, `ls` cannot tell
  - cannot reference a file outside its own file system
  - cannot reference a directory
  - when one of the hard links is deleted, file continue to exist
- Symbolic Links
  - A special file type contains a text pointer referencing either a file or a directory
  - write to a symbolic link, the original file is written
  - delete the symbolic link, the original file still exists
  - delete the original file, the symbolic link is broken

3.6.2 `ln` Command:

- Create a hard link for file: `ln file link`
- Create a symbolic link for an item (directory or file): `ln -s item link`

Try the following commands:

`cp ../test.txt .`	Copy `test.txt` to the current directory
`ln test.txt testLinkH.txt`	Create a hark link
`ln -s test.txt testLinkS.txt`	Create a symbolic link
`ls -l`	Take a look: link number increases
`rm test.txt`	Remove the original file
`ls -l`	Take a look: symbolic link is broken
`less testLinkH.txt`	Page display the hardlink
`less testLinkS.txt`	Page display the symbolic link, not working

12

4. Working with Commands

4.1 Introduction

Commands could be:

- An executable program
- A command built into the shell itself, e.g. shell builtins
- A shell function, e.g. shell script
- An alias

Commands about commands:

- Display a command's type: `type`
- Display which executable program will be executed: `which`
- Get help for **shell builtins**: `help`
- Display a command's **manual page**: `man`
- Display appropriate commands based on **manual page**: `apropos`
- Display a command's **info** (**GNU utilities**): `info`
- Display one-line **manual page** descriptions: `whatis`
- Create an alias for a command: `alias`

Try the following commands:

```
type cd
type ls
which ls
help cd
cd  --help
man cd
man ls
apropos unzip
info ls
whatis ls
alias myAliasCmd='cd ..; ls; cd -'
myAliasCmd
alias ll='ls -l'
alias la='ls -A'
```

13

`myAliasCmd` goes to the parent of the current working directory, lists its content, and comes back to the current working directory. Use `unalias myAliasCmd` to remove an alias.

5 Redirection

5.1 Standard Input, Output, and Error

- Standard Input
  stdin - input by default attached to the keyboard
- Standard Output
  stdout - program result, normally linked to screen
- Standard Error
  stderr - status and error messages that tell us how the program is getting along, linked to screen

Try the following commands, note that `/badDir` is a non-existent directory that generates error message to stderr while the first `ls` generates result to stdout, by default both linked to screen.

```
ls -l /usr/bin
ls -l /badDir
```
14

5.2 Redirecting stdout and stderr

- Redirecting Standard Output to Files
  Output rewrite to a file: `> file`
  Output append to a file : `>> file`
- Redirecting Standard Error to Files
  Write error message to file: `2> file`
  File descriptor 0 indicates input, 1 output, and 2 error
- Redirecting Standard Output and Standard Error to the same one File
  Redirect stdout to file and then redirect stderr to stdout: `> file 2>&1`
  Rewrite both standard output and error to file: `&> file`
  Append both standard output and error to file: `&>> file`
- Disposing of Unwanted Output
  redirect to a special file `/dev/null`: `2> /dev/null`
  `/dev/null` is referred to as bit bucket, receives input but does nothing

Try the following commands and understand the generated content in txt files at each step, you can also use `ls -l` to check the size changes of generated txt files each step:

`ls -l /usr/bin > ls-stdout.txt`	stdout to file
`ls -l /badDir > ls-stdout-badDir.txt`	No result to stdout, error msg is in stderr
`ls -l . >> ls-stdout.txt`	Append result
`ls -l /badDir 2> ls-stderr.txt`	stderror to file
`ls -l /usr/bin /badDir > ls-combined.txt 2>&1`	Both stdout and stderror to the same txt file
`ls -l /usr/bin /badDir &> ls-combined2.txt`	Same as above
`ls -l . &>> ls-combined.txt`	Append stdout and stderror to the txt file
`> empty.txt`	Create an empty text file

15

5.3 Redirecting stdin

Redirect standard input using `cat` command

- `cat` reads one or more files and copies them to standard output

Try the following commands:

```
cat ls-stdout.txt
clear
cat ls-stderr.txt ../test.txt
cat ls-stderr.txt ../test.txt > cat-combined.txt
```
16

- if `cat` is not given any arguments, it reads from standard input (keyboard), ends with ctrl+D, and displays to standard output

Try the following commands:

`cat`	reads from keyboard, displays to screen
`cat > input.txt`	reads from keyboard, writes to `input.txt`
`cat < input.txt`	same as `cat input.txt`, reads from `input.txt`, displays to screen

17

5.4 Pipeline

5.4.1 Pipe operator |

Pipe stdout of one command into stdin of another: *Command1* | *Command2*
Multiple commands can be put together using | operators to form a pipeline.
Try the following command, passing *ls* results to *less* for display:

```
ls -l  /usr/bin | less
```
18

5.4.2 Difference between > and |

- > redirection operator, connects a command with a file:
  Command1 > *file*

- | pipe operator, connects output of one command with the input of the second command:
  Command1 | *Command2*

- Don't do the following, really bad:
  Command1 > *Command2*
  The output of *Command1* overwrites *Command2* program file

5.5 More Commands

- Reads one or more files and copies them to standard output: *cat*
- Sort lines (by default alphabetical ascending) *sort*
- Report or omit repeated lines: *uniq*
- Find text patterns within files: *grep*
- Display the number of lines, words, and bytes in files: *wc*
- Output the first part of a file, use -n to specify # of lines: *head*
- Output the last part of a file, use -n to specify # of lines: *tail*
- Read from standard input and write to standard output and files: *tee*

Multiple commands are piped. Try them and understand what they do:

`ls /bin /usr/bin	sort	less`	List two directories, sort results into one list and display
`ls /bin /usr/bin	sort	uniq	less`	Similar to above, but remove redundant items
`wc ls-stdout.txt`	Show # of lines, words and bytes of ls-stdout.txt file
`ls /bin /usr/bin	sort	wc -l`	Items under two directories are sorted, and counted
`ls /bin /usr/bin	sort	uniq	wc -l`	Count unique items, ignore duplicated ones
`ls /bin /usr/bin	sort	uniq	grep zip`	Find *zip* related items under the two directories
`head -n 5 ls-stdout.txt`	Display the first 5 lines of *ls-stdout.txt* file
`tail -n 5 ls-stdout.txt`	Display the last 5 lines of *ls-stdout.txt* file
`ls /usr/bin	tail -n 5`	Display the last 5 items under */usr/bin*
`ls /usr/bin	tee ls.txt	grep zip`	Write *ls* results to *ls.txt* also pipe to *grep*
`ls`	Use *ls* to check *ls.txt* is generated
`cat ls.txt`	Display *ls.txt* content

19