



# Shells

Gökçe Aydos

This work is licensed under [CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)  

# Lecture

# Goals

- ▶ be able to use Unix- and program-shells
- ▶ know important Unix commands
- ▶ be able to create basic shell scripts

# Theme

- ▶ OS is like a bureaucratic organization
- ▶ request - response
- ▶ requests must obey a specific language
  - ▶ not very appealing to use but powerful (e.g., `cd /; rm -rf *` - we will discover later)
- ▶ *shell* is a program which enables communication between user and a program — *not only OS*

# Shell characteristics

- ▶ dialog-based (request-response) similar to GUI
- ▶ shows a protocol of your dialog
  - ▶ like two persons writing to a notebook

# Shell characteristics - demo

open up a shell, try the commands

- ▶ `ls`
- ▶ `date`
- ▶ `cal`
- ▶ `cowsay hello`

# Dialog

How does the shell establish a dialog?

1. shell shows a *prompt* with a blinking cursor. "Sir, how can I help you?"
2. user types a *command*
3. shell runs the command
4. shell returns back to 1.



# Many kinds of shells

The shells<sup>1</sup> are not only for OS but for every program, e.g.,:

- ▶ Unix-kernel: `sh`, `bash`, `zsh`
- ▶ Windows-kernel: `command.com`, `powershell`
- ▶ shells to remote computers: `ftp`, `ssh`
- ▶ shells to databases: `sqlite3`, `sql`
- ▶ interactive Python: `ipython`

---

<sup>1</sup>remember shell-kernel metaphor. Now we see shell generally as a dialog interface to a program

bash

```
[goekce@joan iti]$
```

"Hi user goekce! You are on the machine joan. You are currently in folder iti. What do you wish?"

# ipython

```
$ ipython
```

```
Python ...
```

```
Type 'copyright', 'credits' or 'license' for more informati
```

```
IPython ... -- An enhanced Interactive Python. Type '?' for
```

```
In [1]:
```

# Starting a shell

- ▶ on the OS search for *terminal* or *shell* and start it
- ▶ the computational environment *Jupyter* also provides a shell
- ▶ here we will use the Unix shell

## Using a shell

type a command which the shell understands and press the `return` key

# Tab completion

- ▶ shells are capable of some empathy
- ▶ if you only enter part of the command and then press the tab key, shell will give you some suggestions

## Tab completion - demo

- ▶ try the commands `ls`, `pwd`, `exit`
- ▶ type them partly and press tab, e.g., type `pw`, then press the tab key
- ▶ what happens?

# Command options

- ▶ commands can have options
- ▶ generally these options begin with a minus symbol - or --
- ▶ how do you know the options?



# Command parameters

- ▶ commands can have parameters:

```
$ ls *.pdf  
iti_shell.pdf iti_operating_systems.pdf
```

- ▶ option vs argument vs parameter

## Getting help

- ▶ use `tab` for the supported options
- ▶ use `-h` or `--help` option
- ▶ use `man command`
- ▶ sometimes an internet search is faster

## Getting help - demo

try `tab`, `-h`, `--help`, `man`, and internet

# File management

- ▶ prompt shows the current directory. All the relative paths you type are relative to your current directory.
- ▶ `pwd` print working directory
- ▶ `cd` change directory

# Listing folder contents

- ▶ `ls` stands for *list*
- ▶ `ls -l` detailed listing
- ▶ `ls *.html` only specific files

## Listing file contents - cat

cat concatenate

file chapter1.md:

```
# A star was born
```

```
There was a cat named Lilly ...
```

file chapter2.md:

```
# First encounter with internet
```

```
Lilly saw other cats becoming famous on ...
```

```
cat chapter1.md chapter2.md
```

But cat is mostly used to see the contents of a single file.

## Listing long file contents

- ▶ `less` (updated version of `more`) for paging long contents
- ▶ you can also use a text editor: `nano`

## Listing parts of files

- ▶ `head` show first lines of the file
- ▶ `head -50` show the first 50 lines
- ▶ `tail ...`



# Managing files

- ▶ `rm` remove (no recycle bin!)
  - ▶ `rm folder-name` won't work
  - ▶ `rm -r` remove recursively (for folders)
- ▶ `cp` copy
- ▶ `mv` move also used for *renaming*

## Discussion - rm

what do the following commands do:

```
cd /
```

```
rm -r *
```

# Creating & deleting folders

- ▶ `mkdir` make directory
- ▶ `rmdir` remove empty directory

## Discussion - rmdir

when could `rmdir` be useful if we have `rm -r`?

## File permission mgmt.

- ▶ `chmod` change access mode
- ▶ requires two parameters:
  1. permission change, e.g., `o+rw`  
u,g,o plus +, - plus r,w,x
  2. file

e.g.:

```
chmod o-r secret.txt
```

## File permission mgmt. - ex.

you want to share `notes.txt` in your home folder with other students. Which command/s would you use?

## Saying/printing

- ▶ echo command:

```
echo hello world!
```

- ▶ looks not useful? Stay tuned for shell programming!

# Downloading

► wget or curl

```
wget duckgo.com
```

```
wget ftp://ftp.ncbi.nlm.nih.gov/pub/pmc/readme.txt
```



# Counting words

- ▶ `wc` word count
- ▶ some Unix commands like to do more than they suggest. `wc` prints newline, word, and byte counts.
- ▶ `wc -l` only line count
- ▶ `wc -w` only word count

```
$ wc -w diary.txt  
68948
```

## Showing differences

- ▶ `diff` difference
- ▶ shows differences between two files

```
$ diff diary.txt diary.txt-backup  
1d0
```

```
< I am very grateful that I met my students today.
```

## Search for file contents

- ▶ `grep` global search for a regular expression and print out matched lines
- ▶ searching in a file

```
$ grep grateful diary.txt
```

```
I am very grateful that I met my students today.
```

```
I was grateful for their empathic conversation today
```

- ▶ searching recursively in a folder: `grep -r`
- ▶ searching case-insensitive `grep -i`

# Shell programming

- ▶ shell supports programming by, e.g.:
  - ▶ for, while loops
  - ▶ connecting the inputs & outputs of different commands

# Output Redirection

- ▶ some commands print directly to the shell
- ▶ using `>` you can redirect an output to a *file*
  - ▶ `>` overwrites
  - ▶ `>>` appends

```
$ ls
diary.txt iti-exercise1.pdf
$ ls > directory-listing
$ cat directory-listing
diary.txt iti-exercise1.pdf
```

# Input redirection

► < reads from a file instead of interactive input

```
$ cat hello.txt
```

```
hello world!
```

```
$ wc hello.txt
```

```
1  2 13 hello.txt
```

```
$ wc < hello.txt
```

```
1  2 13
```

# Piping

- ▶ using `|` we can forward the output of a command to the input of another command
- ▶ called *piping*

```
$ ls  
diary.txt iti-exercise1.pdf  
$ ls | wc -w  
2
```

# Shell scripts

- ▶ *shell scripts* are shell command sequences which we can reuse
- ▶ steps:
  1. we store the commands in a file
  2. we run it using `sh my-script.sh`



## Shell script - example

```
$ cat show-info-about-this-folder.sh  
echo Current folder:  
pwd  
echo contains this number of files and folders:  
ls | wc -w
```

## Shell script - example II

```
$ sh show-info-about-this-folder.sh
```

```
Current folder:
```

```
/home/goekce/iti
```

```
contains this number of files and folders:
```

```
2
```

## Parameters for shell scripts

we can supply positional parameters to our scripts using \$1, \$2, \$@ etc.

```
$ cat backup.sh  
echo Making a backup of $1  
cp $1 $1.backup
```

## Parameters for shell scripts II

```
$ sh backup.sh diary.txt  
$ ls diary.txt*  
diary.txt diary.txt.backup
```

## Tips for using last used commands

- ▶ backwards in your history
  - ▶ use the up-arrow key on your keyboard, repeat it if needed
- ▶ searching backwards in your history
  - ▶ press `Ctrl+r`, then start typing a command that you used
  - ▶ press `Ctrl+r` until you find what you search for

# Summary

- ▶ shells establish a dialog between a user and a program
- ▶ `cd`, `pwd`, `ls`, `cat`, `less`, `head`, `tail`, `rm`, `cp`, `mv`, `chmod`, `echo`, `wget`, `grep`, `wc`, `diff`
- ▶ redirection `>`, `<`, and piping `|`
- ▶ shell scripts