The Unix Shell

10 DIM FourDigitPassword INTEGER
20 FOR i = 0 TO 9
30 FOR j = 0 TO 9
40 FOR k = 0 TO 9
50 FOR l = 0 TO 9
60 FourDigitPassword = getFourDigits (i,j,k,l)
70 IF checkPasswordMatch(FourDigitPassword) = TRUE TH
80 GOTO 140
90 END
100 NEXT l
110 NEXT k
120 NEXT j
130 NEXT i
140 PRINT FourDigit

```
 _____________________________________
/ Welcome to the thrilling world of the \
\ Unix shell                            /
 -------------------------------------
        \   ^__^
         \  (@@)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

```
$ cowsay "Welcome to the thrilling world of the Unix shell"
```

Overview

1. Shell, what/why?
2. Filesystem
3. The Unix Philosophy
4. Commands
5. Permissions
6. Streams & Pipelines
7. CLI Text Editors

Learning Objectives

- List some common UNIX CLI tools
- Solve some simple problems using CLI tools
- Describe the UNIX philosophy

Shells have been around almost since the dawn of computing.

ASCII Star Wars Episode IV:

```
telnet towel.blinkenlights.nl
```

Also, you can play Tetris online:

```
ssh netris.rocketnine.space
```

What is a shell?

A shell is a text-based interface with your operating system

A shell can do the same things your GUI environment does:

- Launch applications
- Open, edit and manage files and folders
- Send email
- Code
- Play Star Wars Episode IV

The Shell

- Bash is the default shell in most Unix OSs
- MacOS now uses Zsh (used to be bash)
- Windows has `cmd` and Powershell
- Windows can also run `git bash`, as well as `WSL`
- Others exist but are less common
- Shells come pre-installed with each and every OS

Reasons why you should use a shell

Installed everywhere: Desktops, servers, RPis... GUI environments are not

Lightweight: GUI environments have high resource demands (memory, CPU etc.)

Lower bandwidth: Uses much less than a GUI, hence are often the default when administering remote/cloud servers.

Concise and powerful: Designed for automation

Getting Around

Open your terminal and play along!

Simple commands

`pwd`: print the current directory

`ls`: list the files in the current directory

`cd`: change the current directory

`cat`: print the contents of a file

Quiz Time! 🤓

What is a shell?

1. A graphical-based interface within your operation system
2. A text-based interface stored on your hard drive
3. A desktop environment within your operating system
4. A text-based interface within your operating system
5. Commonly found on the beach

Answer: 4

Filesystem Hierarchy

A filesystem is the system used by your OS to store and manage all of your files.

Filesystems are usually hierarchical.

In macOS and Linux, your filesystem is a tree and the top of the tree is known as the root directory, `/`.

Unix Filesystem

Key Directories

- `/`: your filesystem starts here
- `/home`: personal files
- `/etc`: system config files
- `/bin`: applications

Paths

The location of a file in your computer is given by its path

Paths can be:

- Absolute: always start at the root directory, e.g. `/home/dennis/code/src/hello.txt`
- Relative: based on your current directory, e.g. (assuming I'm already at `/home/dennis`) `code/src/hello.txt`

Building Relative Paths

. current directory

. . parent directory

You can chain them

```
$ pwd
/home/dennis/code
$ ls .
$ cd ./src
$ cat ./hello.txt.py
$ ls ../../Pictures/cats
```

Exercise

Launch a terminal, switch to the directory where your mini-project code is located and print the contents of a source code file to the screen

The Unix philosophy

The Unix philosophy was created by the original developers of Unix and its associated tools.

It was documented and published in 1978, but is still very relevant to how we want to be developing software today!

The Unix philosophy

Make each program do one thing well. To do a new job, build afresh rather than complicate old programs by adding new "features".

The Unix philosophy

Expect the output of every program to become the input to another, as yet unknown, program.

Don't clutter output with extraneous information.

Avoid stringently columnar or binary input formats.

Don't insist on interactive input.

The Unix philosophy

Design and build software, even operating systems, to be tried early, ideally within weeks.

Don't hesitate to throw away the clumsy parts and rebuild them.

The Unix philosophy

Use tools in preference to unskilled help to lighten a programming task, even if you have to detour to build the tools and expect to throw some of them out after you've finished using them.

The Anatomy of a Shell Command

Shell commands can take any number of flags and arguments, or none at all

Example

```
$ ls -a -l /home/dennis/code
# Can be abbreviated as
$ ls -al /home/dennis/code
```

List the contents of the `/home/dennis/code` directory with details (-l), including hidden files (-a). Flags can be concatenated

Where are all the flags, man?

`man` will display the help page for a command

For example, to see the help page for the `ls` command we just looked at:

```
$ man ls
```

Quiz Time! 🤓

What does the `cd` command do?

1. Print out the contents of a file
2. Print the current file path
3. Change to a different directory
4. Print the contents of a folder

Answer: 3

Exercise

List your directory sorted by timestamp.

Remember to use the `man` page for the command to find out the arguments you need.

Filesystem Commands

`touch`: create an empty file

`mv`: move/rename a file or directory

`cp`: copy a file to a new location

`rm`: remove a file

`mkdir`: create a directory

`rmdir`: remove an empty directory

Examples

```
$ mkdir folder-one
$ mkdir folder-two
$ mkdir backups
$ touch folder-one/hello.txt
$ mv folder-one/hello.txt folder-two
$ cp folder-two/hello.txt backups/

# Deletes a single file
$ rm folder-two/hello.txt

# Deletes an entire folder and all its sub-folders and files
$ rm -rf folder-two
```

More Useful Commands

`echo`: print some text to the screen

`history`: show list of previously run commands

`clear`: wipe the screen

`less`: show the contents of a file in a paginated way

`head`: display the first `n` lines of input

`tail`: display the last `n` lines of input

Exercise

In your shell, create a new "backups" directory and copy your entire mini-project directory (and files) into it

Searching

Finding text

`grep`: search text in files

```
$ grep "hello" ./folder/*
```

A couple of useful flags are -i (case-insensitive) and -r (recursive)

```
$ grep -ri "hello" ./folder
```

Finding files

`find`: search a file by name or other properties

```
$ find ./ -name "app.py"
# You can also use wildcards
$ find ./ -name "app*"
```

Finding programs

`whereis`: get the location of a program

```
$ whereis grep
```

Globbing

Globbing allows you to refer to many files without the hassle of typing out their paths in full

The question mark (`?`) matches any one character in a path

```
$ ls ./Pictures/longcat.jp?g
longcat.jpeg
```

The asterisk (`*`), a.k.a. wildcard, matches every letter in a path, or no letters

```
$ ls ./Pictures/longcat.jp*g
longcat.jpg longcat.jpeg
```

```
$ ls ./Pictures/cats/*
longcat.jpg longcat.jpeg cat.jpg anothercat.png
```

Both wildcard characters can be put anywhere in a path

```
$ ls ./code/miniproject/*.py
app.py utils.py text.py

$ ls ./code/miniproject/app.*
app.py
```

Quiz Time! 🤓

What does `grep` do?

1. Searches for the location of a file
2. Searches text in files
3. Searches for files in a file system
4. Searches for the location of a folder

Answer: `2`

Exercise

Create a sensible directory structure for your mini-project using the shell

Example:

```
miniproject
├── documentation
│   ├── overview.md
│   └── tea-types.md
├── README.md
├── source
│   ├── app.py
│   ├── beer.py
│   ├── coffee.py
│   └── tea.py
└── tests
    ├── app\_test.py
    └── tea\_test.py
```

Permissions

Unix-like systems allow you to control who has access to files in a computer and what they're allowed to do

- Users
- Groups
- Everyone

We can restrict a user's ability to read, write files or run programs

For example if we ran the command `ls -la` we would see;

```
total 24
drwxrwxr-x.   3 jerry jerry  4096 Aug 24 17:29 .
drwxr-xr-x. 136 jerry jerry 12288 Aug 29 15:18 ..
drwxr-xr-x.   4 jerry devs   4096 Aug 19 14:43 academy-revealjs
-rw-r--r--.   1 jerry jerry    51 Aug 24 17:29 file.py
```

This is now showing us the parent folders `..` and showing us the permissions currently set on each file system object.

Setting permissions with chmod

```
$ chmod < target >< +/- >< action >
```

```
# Everyone can read
$ chmod a+r hello.txt
# Only the owner of the file can write
$ chmod u+w hello.txt
# Everyone can no longer read
$ chmod a-r hello.txt
```

Targets:

- u: owner
- g: group
- e: everyone

Access type:

- r: read
- w: write
- x: execute

Setting permissions the hard way

This is worth mentioning as you'll come across it many times in your career, so it's important to also understand how it works.

Permissions can be expressed numerically too, as three groups of octets (0-7), one per target.

Each octet represents the level of access given to the resource.

#	Permission	rwx	Binary
7	read, write, execute	rwx	111
6	read, write	rw-	110
5	read, execute	r-x	101
4	read only	r--	100
3	write, execute	-wx	011
2	write only	-w-	010
1	execute only	--x	001
0	none	---	000

Examples

All permissions for everyone

```
$ chmod 777 file.py
```

Owner can do anything, no-one else has access to file

```
$ chmod 700 file.py
```

Owner can read and write, group can read and no one else can access the file

```
$ chmod 640 file.py
```

Quiz Time! 🤓

I want to set the following permissions for `file.py` - owner/user can do everything, group can read and execute, everyone else can read only - which is the correct command?

1. `$ chmod 732 file.py`
2. `$ chmod 754 file.py`
3. `$ chmod 650 file.py`
4. `$ chmod 444 file.py`

Answer: `2`

Quiz Time! 🤓

I want to set the following permissions for `file.py` - owner/user and group should be able to read and execute, everyone else has no permissions - which is the correct command?

1. `$ chmod 661 file.py`
2. `$ chmod 777 file.py`
3. `$ chmod 650 file.py`
4. `$ chmod 550 file.py`

Answer: `4`

Regular users cannot modify protected system resources such as external devices (/dev), system commands or system directories (any immediate children of the root directory, except /home and maybe /var, /tmp)

The Root User

There are two types of users in a Unix-like environment:

- Regular users: can only create and modify files and programs they own or they've been given permission to
- The superuser: a.k.a. root, administrator, etc., has full control over the entire OS

With great power comes great responsibility

You can log in as the root user by typing `su` in a terminal.

```
$ su
Password:
# echo "Congratulations, you're now root!"
```

You should avoid logging into your system as root. However, sometimes we need to make changes for which we require elevated privilege

The sudo command

The sudo command

`sudo` allows regular users to become root temporarily so they can run privileged commands

```
$ cp /bin/python3 /bin/python-three
cp: cannot create regular file 'python-three': \
  Permission denied
$ sudo cp /bin/python3 /bin/python-three
```

sudo's and sudont's

The risk with running the root account, security aside, is that you may accidentally run a harmful command and end up destroying something important

- Use root account as least as possible.
- Protect root account with a strong password
- Use sudo with caution

Passing information along

Applications receive input data to work on and produce some output when they're done

But where does the data come from and where does it go?

The shell provides 3 separate channels to enable data flow in and out of applications

These channels are known as standard streams and they are connected to different things by default

Standard Streams

- Input: `stdin`
- Output: `stdout`
- Errors: `stderr`

Text terminal
Keyboard
#0 stdin
Process
#1 stdout
Display
#2 stderr

You can attach the standard streams to different things like files or peripherals

For example, you can ask a program to send its output to a file instead of the screen with the redirect operator

```
# Send today's fortune to a file instead of printing it to the screen
$ fortune > fortune-today.txt
```

Unix Pipe

We've been doing this manually so far, but we can also get programs to talk to each other

We can redirect the output from one program as input to another program with pipes

Pipelines

You can join up many Unix pipes to construct complex pipelines that filter down and transform your data before producing a final result, e.g.:

```
sort colours.txt | uniq -c | sort -r | head -3
```

Create the `colours.txt` file first, then try it out!

Package managers

Package managers let you install and keep track of software on your machine

macOS uses homebrew

Every version of Linux has its own package manager

Ubuntu uses APT

Ubuntu Package Manager

```
# If you don't know the name of a package, look it up
$ apt-cache search cowsay
# Update your package information
$ sudo apt update
# Install a package
$ sudo apt install cowsay
# Uninstall a package
$ sudo apt remove cowsay
# Update all installed packages
$ sudo apt upgrade
```

macOS homebrew package manager

```
# If you don't know the name of a package, look it up
$ brew search cowsay
# Update your package information
$ brew update
# Install a package
$ brew install cowsay
# Uninstall a package
$ brew remove cowsay
# Update all installed packages
$ brew upgrade
```

Exercise

Install "fortune" and "cowsay" on your machine and have the cow say the "fortune" message, then uninstall them (if you want)

Text Editors

nano

nano is a simple and easy to learn command-line text editor

Try it out with `nano`

vim

vim is powerful

vim is everywhere

vim is hard

Try it out with `vi`

If you **really** want to learn it, just run `vimtutor` in your shell

Figure out how to exit vim, it isn't easy!

Things you can do in the shell

```
# Order a list of Python source files by line count,
# sorting them in descending order
find src -name "*.py" | xargs wc -l | sort -r
# Find and replace a string across any number of files
# and directories specified
grep -rli 'dog' * | xargs -i@ sed -e 's/dog/cat/g' @
```

Automation in the Shell

Shell scripting allows you to write shell "programs" to automate complex tasks.

Shell scripts are written in the Bash scripting language.

The Bash scripting language contains all the basic building blocks you'd expect in a regular programming language.

Example Bash File

Instructor to distribute `example-shell-script.sh`

Your shell config file

The only shell script you should care about for now is `.bashrc` (or `.zshrc`)

This is your shell config file, which is loaded every time you open a new terminal

Your shell config file lives in your home directory, `/home/john` on Linux or `/User/john` on macOS

You can define variables in this file and export them so that they're available in your shells

Environment Variables

Global variables available in your shells

```
# Print the value of an environment variable
echo $PATH
# Set an environment variable
export MYNEWVAR="hello"
```

Exercise

Locate your .bashrc/.zshrc file and create a new environment variable like this:

```
export EDITOR=nano
```

Learning Objectives Revisited

- List some common UNIX CLI tools
- Solve some simple problems using CLI tools
- Describe the UNIX philosophy

Terms and Definitions Recap

Shell: A computer program which exposes an operating system's services to a human user or other program.

Unix: A family of multitasking, multi-user computer operating systems that derive from the original AT&T Unix, development starting in the 1970s at the Bell Labs research center.

Glob: Glob patterns specify sets of filenames with wildcard characters.

Terms and Definitions Recap

Wildcard: A symbol used to replace or represent one or more characters.

Standard Stream: Interconnected input and output communication channels.

Package Manager: A collection of software tools that automates the process of installing, upgrading, configuring, and removing computer programs for a computer's operating system in a consistent manner.

Further Reading

- Text handling tools: sed, awk, cut, tr, uniq, sort, perl
- Archiving and compressing: tar, zip, gzip, bzip2
- Great video of Unix being used in the 80s with interviews from it's primary developers

Further Reference and Credits

- Pipeline (pic)
- Pipe (pic)
- Standard streams (pic)