

# Introduction to Unix

Victor Eijkhout

2020



# Justification

Unix, in particular Linux, is the *de facto* operating system in High-Performance Computing (HPC).

# Files and such

# ls, touch

- List files: `ls`
- Maybe your account is still empty: do `touch newfile`, then `ls` again.
- Options: `ls -l` or for specific file `ls -l newfile`.

# Display / add to file: cat

- Display a file: `cat myfile`
- Put something in a file: `cat > myfile`  
end with Control-D.  
Or use an editor, but this is sometimes still useful.
- Now `cat` it again.

## **cp, mv, rm**

- **Copy:** `cp file1 file2`  
Do this, check that it's indeed a copy.
- **Rename or 'move':** `mv file1 file2`  
check that the original file doesn't exist any more.
- **Remove:** `rm myfile`  
This is irrevocable!

# Dealing with large (text) files

- If a file is larger than your screen:

```
less yourfile
```

- If the start or end is interesting enough:

```
head yourfile, tail yourfile
```

- Explore options: `head -n 5 yourfile`



# Directories

# Directories

- Make a subdirectory 'folder': `mkdir newdir`
- Check where you are: `pwd`
- Now go to the new directory: `cd newdir` and `pwd`  
'change directory' and 'present working directory'
- Back to your home directory: `cd` without further arguments.

# Paths

- Do:
  1. `cd newdir`
  2. `touch nested_file`
  3. `cd`
- Now: `ls newdir/nested_file`
- That is called a path
  - Relative path: does not start with slash
  - Absolute path (such as `pwd` output): starts at root

# More paths

- Path to your home directory: tilde `cd ~`
- Going out of a directory: `cd ..`
- You can use this in paths: `ls newdir/subdir1/../subdir2`

Exercise: copy the lorem ipsum file from the repo to a new directory.

# Permissions

# Basic permissions

- Three degrees of access: user/group/other
- three types of access: read/write/execute

<i>user</i>	<i>group</i>	<i>other</i>
<i>rwX</i>	<i>rwX</i>	<i>rwX</i>

Example: `rw-r-----:`

owner read-write, group read, world: nothing

# Permission setting

- Add permissions `chmod g+w myfile`
- recursively: `chmod -R o-r mydirectory`
- Permissions are a binary number: `chmod 644 myfile`

# Share files

- Make a file in your `$WORK` file system, and make it visible to the world.
- Ask a fellow student to view it.
- $\Rightarrow$  also necessary to make `$WORK` readable.  
(Not a good idea to make `$HOME` readable.)



# The x bit

The x bit has two meanings:

- For regular files: executable.
- For directories: you can go into them.
- Make all directories viewable:

```
chmod -R g+X,o+X rootdir
```

# Redirection, pipes

# In/Output redirection

- There are three standard files: `stdin`, `stdout`, `stderr`
- Normally connected to keyboard and screen.

- Redirection: standard out to file:

```
ls > directorycontents
```

(actually, screen is a file, so it is really a **redirect**)

- Standard in from file: `mail < myfile`  
(actually, the keyboard is also a file, so again **redirection**)

Exercise: make a copy of a file, using redirection (so no `cp` command).

# Pipes

- Redirection is command-to-file.

- Pipe: command-to-command

```
ls | wc -l
```

- Unix philosophy: small building blocks, put together.

## More command sequencing

More complicated case of one command providing input for another:

```
echo The line count is wc -l foo
```

where `foo` is the name of an existing file.

Use backquotes:

```
echo The line count is `wc -l foo`
```

Exercise: this way `wc` prints the file name. Can you figure out a way to prevent that from happening?

# Shell programming

# Command execution

- Some shell commands are built-in, however most are programs.
- `which ls`
- Exercise: what can you find out about the `ls` program?
- Programs can be called directly: `/bin/ls` or found along the search path `$PATH`:  
`echo $PATH`

# Things that look like commands

- Use `alias` to give a new name to a command:  
`alias ls='ls -F'`  
`alias rm='rm -i'`
- There is a shell level `function` mechanism, not explained here.



# Processes

<code>ps</code>	list (all) processes
<code>kill</code>	kill a process
<code>CTRL-c</code>	kill the foreground job
<code>CTRL-z</code>	suspect the foreground job
<code>jobs</code>	give the status of all jobs
<code>fg</code>	bring the last suspended job to the foreground
<code>fg %3</code>	bring a specific job to the foreground
<code>bg</code>	run the last suspended job in the background

Exercise:

how many programs do you have running?

# Variables

- `PATH` is a variable, built-in to the shell
- you can make your own variables:

```
a=5
```

```
echo $a
```

No spaces around the equals!

Exercise: what happens when you try to add two variables together?

```
a=3
```

```
b=5
```

# Variable manipulation

- Often you want to strip prefixes or suffixes from a variable:

`program.c`  $\Rightarrow$  `program`

`/usr/bin/program`  $\Rightarrow$  `program`

- Parameter expansion:

`a=program.c`

`echo ${a%.c}`

`a=/foo/bar/program.c`

`echo ${a##*/}`

# Conditionals

- Mostly text-based tests:

```
if [ $a = "foo" ] ; then
    echo "that was foo"
else
    echo "that was $a"
fi
```

- Single line:

```
if [ $a = "foo" ] ; then echo "foo" ; else echo "something" ; fi
```

Note the semicolons!

also spaces around square brackets.

# Other conditionals

- Numerical tests:/

```
if [ $a -gt 2 ] ....
```

- File and directory:

```
if [ -f $HOME ] ; then echo "exists" ; else echo "no such" ; fi  
if [ -d $HOME ] ; then echo "directory!" ; else echo "file" ; fi
```

# Looping

- Loop: for item in list  
the item is available as macro  

```
for letter in a b c ; do echo $letter ; done
```
- Loop over files:  

```
for file in * ; do echo $file ; done
```

Exercise: loop through your files, print which ones are directories.

Exercise: for each C program, remove the object file.

# Numerical looping

- Type `seq 1 5`
- Exercise: can you figure out how to loop 1...5?

```
n=12
## input
for i in ..... ; do echo $i ; done
## output
1
....
12
```

# Scripting



# Script execution

- Create a script `script.sh`:

```
#!/bin/bash  
echo foo
```

- Can you execute this? Does the error suggest a remedy?
- What is the remaining problem?

# Arguments

- You want to call `./script.sh myfile`
- Parameters are `$1` et cetera:  

```
#!/bin/bash  
echo "$1 is a file"
```
- How many arguments: `$#`

# Exercise

Write a script that takes as input a file name argument, and reports how many lines are in that file.

Edit your script to test whether the file has less than 10 lines (use the `foo -lt bar test`), and if it does, `cat` the file. Hint: you need to use backquotes inside the test.

Add a test to your script so that it will give a helpful message if you call it without any arguments.