Systems Programming

Lecture 3: Bash scripting in UNIX

Stuart James

stuart.a.james@durham.ac.uk

Durham University

Practicals

- Practicals start this week.
- Does everyone know where/when their practical group is?
- If not, contact myself or Stuart as soon as possible!
- Make sure you fully understand the github classroom assignment (<5 minutes if you have a github account already).


Last Lecture

- We looked at UNIX and some very basic commands for navigating around

Today

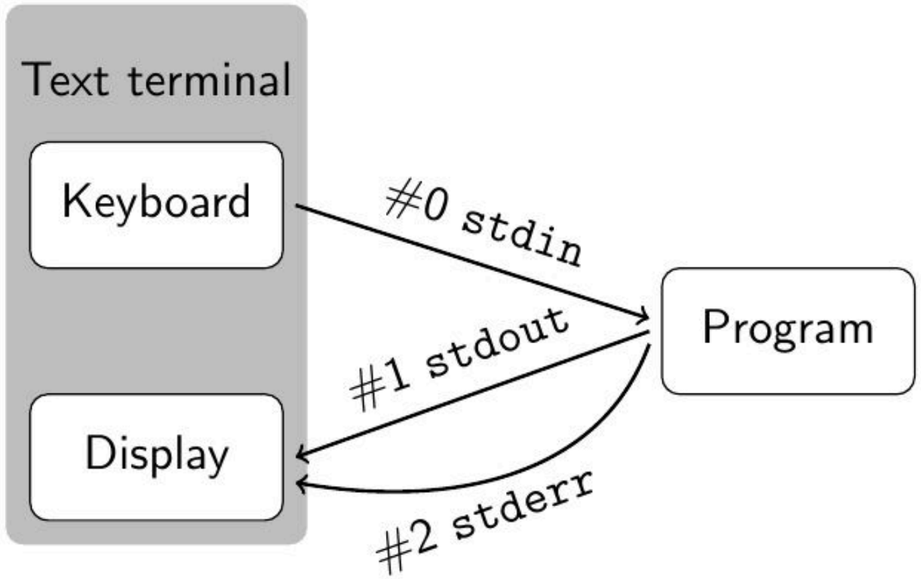
- We will try out some more advanced bash commands and scripting


Recap

https://PollEv.com/stuartjames

Recap: `stdin`, `stdout` and `stderr`

- Remove the need to worry about I/O devices
- Two types of output, each can be redirected



`sort`

- What does it sort?
- A file (if specified)
  - `stdin`: standard input, by default from terminal
- Where does it put the results?
  - `stdout`: standard output, by default the terminal
  - or a file with `-o filename`


`sort`

- Can redirect output to file with `>`
  - e.g. `sort infile.txt > outfile.txt`
- Can redirect input from file with `<`
  - `sort < infile.txt > outfile.txt`
  - or `sort -o outfile.txt infile.txt`


`tr` - translate

- `tr SET1 SET2`
  - translates or deletes characters from `SET1` to `SET2`
  - e.g. `tr 'A-Z' 'a-z'` makes a lower case version of `stdin`
  - option `-c` takes complement of `SET1`
  - option `-s` squeezes repeats to a single character
  - option `-d` deletes all characters in `SET1`
  - e.g. `tr -dc '[:print:]'` - deletes all non printable characters


`tr` - translate

- `tr SET1 SET2`
  - translates or deletes characters from `SET1` to `SET2`
  - e.g. `tr 'A-Z' 'a-z'` makes a lower case version of `stdin`
  - option `-c` takes complement of `SET1`
  - option `-s` squeezes repeats to a single character
  - option `-d` deletes all characters in `SET1`
  - e.g. `tr -dc '[:print:]'` - deletes all non printable characters

In [7]:
```
1 !cat nonprint.txt
```

```
This is some text with non-printable characters: ^@^A^B^C^D^E
```


`tr` - translate

- `tr SET1 SET2`
  - translates or deletes characters from `SET1` to `SET2`
  - e.g. `tr 'A-Z' 'a-z'` makes a lower case version of `stdin`
  - option `-c` takes complement of `SET1`
  - option `-s` squeezes repeats to a single character
  - option `-d` deletes all characters in `SET1`
  - e.g. `tr -dc '[:print:]'` - deletes all non printable characters

In [7]:
```
1 !cat nonprint.txt
```

```
This is some text with non-printable characters: ^@^A^B^C^D^E
```

In [10]:
```
1 !cat nonprint.txt | tr -dc '[:print:]'
```

```
This is some text with non-printable characters: ^@^A^B^C^D^E
```


`tr` - translate

- `tr SET1 SET2`
  - translates or deletes characters from `SET1` to `SET2`
  - e.g. `tr 'A-Z' 'a-z'` makes a lower case version of `stdin`
  - option `-c` takes complement of `SET1`
  - option `-s` squeezes repeats to a single character
  - option `-d` deletes all characters in `SET1`
  - e.g. `tr -dc '[:print:]'` - deletes all non printable characters

In [7]:
```
1 !cat nonprint.txt
```

```
This is some text with non-printable characters: ^@^A^B^C^D^E
```

In [10]:
```
1 !cat nonprint.txt | tr -dc '[:print:]'
```

```
This is some text with non-printable characters: ^@^A^B^C^D^E
```


`uniq`

- Remove or report repeated lines
- Can be used with `sort` to find lines repeated throughout document;
- e.g. `sort | uniq`
- Use `-c` option to count number of repetitions


`uniq`

- Remove or report repeated lines
- Can be used with `sort` to find lines repeated throughout document;
- e.g. `sort | uniq`
- Use `-c` option to count number of repetitions

In [11]:
```
1 !cat file.txt
```

```
!       LEAF    4.5     56      BROWN
@       NEEDLE  3.0     45      SILVER
$       DESK    104.0   453     WHITE
%       CHAIR   56.5    124     MAGNOLIA
@       NEEDLE  3.0     45      SILVER
@       NEEDLE  3.0     45      SILVER
```


`uniq`

- Remove or report repeated lines
- Can be used with `sort` to find lines repeated throughout document;
- e.g. `sort | uniq`
- Use `-c` option to count number of repetitions

In [11]:
```
1 !cat file.txt
```

```
!       LEAF    4.5     56      BROWN
@       NEEDLE  3.0     45      SILVER
$       DESK    104.0   453     WHITE
%       CHAIR   56.5    124     MAGNOLIA
@       NEEDLE  3.0     45      SILVER
@       NEEDLE  3.0     45      SILVER
```

In [12]:
```
1 !tr 'A-Z' 'a-z' < file.txt | tr -cs 'a-z' '\n' | sort | uniq -c | sort -n
```

```
   1
   1 brown
   1 chair
   1 desk
   1 leaf
   1 magnolia
   1 white
```


Defining our own UNIX command

- UNIX commands are just executables.
- Most are written in C.
- Once we have started on the C component we can also use it to write our own commands.


File handling

- Files are stored in a hierarchical structure ; allows grouping
- Navigation (summary from last lecture and some additions)
  - `ls` - list the contents of the current folder
  - `cd` - change folder
  - `mkdir` - make new folder
  - `mv` - move a file / folder (also used to rename)
  - `cp` - copy a file / folder
  - `rm` - delete a file or with -r a directory
  - `du` - how much space does a folder / file take?
  - `find` - list all files


File Permissions in UNIX

- Each file has three types of permissions:
  - **Read (r)**: Allows viewing the contents of the file.
  - **Write (w)**: Allows modifying the contents of the file.
  - **Execute (x)**: Allows running the file as a program.
- **Permission groups**:
  - **Owner**: The user who owns the file.
  - **Group**: Other users who are in the file's group.
  - **Others**: All other users.
- **Permission representation**:
  - Represented as a string of 10 characters, e.g., `-rwxr-xr--`
    - The first character indicates the file type (`-` for regular file, `d` for directory).
    - The next three characters are the owner's permissions.
    - The following three characters are the group's permissions.
    - The last three characters are the others' permissions.


File Permissions in UNIX

- **Permission representation**:
  - Represented as a string of 10 characters, e.g., `-rwxr-xr--`
    - The first character indicates the file type (`-` for regular file, `d` for directory).
    - The next three characters are the owner's permissions.
    - The following three characters are the group's permissions.
    - The last three characters are the others' permissions.


File Permissions in UNIX

- **Changing permissions**:
  - Use the `chmod` command to change file permissions.
  - Syntax: `chmod [permissions] [file]`
  - Example: `chmod u+x file.sh` (adds execute permission for the owner).


File Permissions in UNIX

- **Changing permissions**:
  - Use the `chmod` command to change file permissions.
  - Syntax: `chmod [permissions] [file]`
  - Example: `chmod u+x file.sh` (adds execute permission for the owner).

In [3]:
```
1 !touch example_file.sh
```


File Permissions in UNIX

- **Changing permissions**:
  - Use the `chmod` command to change file permissions.
  - Syntax: `chmod [permissions] [file]`
  - Example: `chmod u+x file.sh` (adds execute permission for the owner).

In [3]:
```
1 !touch example_file.sh
```

In [4]:
```
1 !ls -l example_file.sh
```

```
-rw-r--r--@ 1 sjames  staff  0 13 Oct 11:19 example_file.sh
```


File Permissions in UNIX

- **Changing permissions:**
  - Use the `chmod` command to change file permissions.
  - Syntax: `chmod [permissions] [file]`
  - Example: `chmod u+x file.sh` (adds execute permission for the owner).

In [3]:
```
!touch example_file.sh
```

In [4]:
```
!ls -l example_file.sh
```

```
-rw-r--r--@ 1 sjames  staff  0 13 Oct 11:19 example_file.sh
```

In [5]:
```
!chmod u+x example_file.sh
!ls -l example_file.sh
```

```
-rwxr--r--@ 1 sjames  staff  0 13 Oct 11:19 example_file.sh
```


Shell scripts

- A Shell Script is simply a collection of commands enclosed in a file.
- Why are they useful?
- Example: a deployed web application (written in Java) needs updating, so
    - Tomcat web-server must be shut down
    - program re-compiled
    - put into a `.jar` file
    - copied to the correct location
    - Tomcat restarted


Shell scripts

- Above example:
  - involves typing in 5 separate commands at the command line
  - not impossible, but it can get rather time-consuming
  - Putting the 5 commands into a shell script enables them to be executed at the command line in one single command


Writing a Shell Script

- You can write shell scripts in any text editor of your choosing.
- They should be saved with a `.sh` extension, e.g. `myscript.sh`.
- They must all begin with the line `#!/bin/bash`
  - "`#!`" tells UNIX this is a script that can be run.
  - `/bin/bash` tells Linux what program to run the script with.


Example

- This script creates a new directory, changes into it and creates two new text files

```
#!/bin/bash
mkdir newDirectory
cd newDirectory
touch file1.txt
touch file2.txt
```


How do you run a shell script?

- Firstly, you need to make sure you have permission to execute the script file Use the `chmod` command to do this
  - `chmod a+x myscript.sh`
- Then, at the command line, type `./scriptname` and your script should run
  - e.g. `./myscript.sh`


For loops

- A handy little tool for doing the same operation to lots of files

```
#!/bin/bash
for f in *
do
  #something in here
  echo $f
done
```


Parameters

- You can add parameters to a script when you run them
- `./myscript.sh foo bar`
  - "`foo`" and "`bar`" are the parameters here
- Refer to them using the `$` sign in scripts
  - `$1`, `$2`, etc.


The `if` statement in shell scripts

```
#!/bin/bash
if [ $1 -lt $2 ]
then
  echo "yes" $1 "is less than" $2
else
  echo "no it isn't"
fi
```

- The `else` bit is optional
- Uses `==`, `!=`, `-gt`, `-lt`, `-le`, `-ge` for equality, inequality, greater than, less than, less than or equal, greater than or equal


Some last bits

- `if [ -e FILE ]` - true if `FILE` exists
- `if [ -z STRING ]` - true if `STRING` is empty
- Variables:
  - `VAR="Hello World"`
  - `echo $VAR`
  - `TD="The time is `date`"`
  - `echo $TD`
    - `The time is Mon 09 Oct 13:44:14 GMT 2023`


Summary

- `stdin`/`stdout`/`stderr` provide hardware independent I/O.
- can redirect input and output.
- Use C to write new programs for UNIX!
- Shell scripts allow you to do more.
- There are:
  - 1000's of commands
  - 10's of shells
  - 10's of scripting languages
- You can do almost anything in the shell


What is git?

- Git is software for:
  - tracking changes in files
  - coordinating work among collaborators
  - with support for CI tools


Short history of git

- 1991–2002: Changes to the linux kernel were passed around as patches and archived files.
- In 2002, the Linux kernel project began using a proprietary DVCS (distributed version control system) called BitKeeper.
- In 2005, BitKeeper's free-of-charge status was revoked
- Thus, the Linux development community (in particular Linus Torvalds) developed git.


Short history of git

- Main goals:
  - Speed
  - Simple design
  - Strong support for non-linear development (thousands of parallel branches)
  - Fully distributed
  - Able to handle large projects like the Linux kernel efficiently (speed and data size)


Git over the command line

Git clone

- Create a copy of a given repository on your machine


Git over the command line

Git clone

- Create a copy of a given repository on your machine

In [9]:
```
!ls
```

```
durhamlogo.png              qr-poll.png
for-quiz                    rise.css
introunix-part1.ipynb       t-and-r.jpg
introunix-part1.slides.html tar.png
lecture2-allfiles.zip       unix-timeline.png
lecture2-slides.pdf
```


Git over the command line

Git clone

- Create a copy of a given repository on your machine

In [9]:
```
!ls
```

```
durhamlogo.png              qr-poll.png
for-quiz                    rise.css
introunix-part1.ipynb       t-and-r.jpg
introunix-part1.slides.html tar.png
lecture2-allfiles.zip       unix-timeline.png
lecture2-slides.pdf
```

In [10]:
```
!git clone https://github.com/COMP2221/example-repository.git
```

```
Cloning into 'example-repository'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 13 (delta 0), reused 7 (delta 0), pack-reused 0
Receiving objects: 100% (13/13), done.
```

In [11]:
```
!ls
```

```
durhamlogo.png              lecture2-slides.pdf
example-repository          qr-poll.png
for-quiz                    rise.css
```


Git over the command line

Git clone

- Create a copy of a given repository on your machine

In [9]:
```
1 !ls
```

```
durhamlogo.png              qr-poll.png
for-quiz                    rise.css
introunix-part1.ipynb       t-and-r.jpg
introunix-part1.slides.html tar.png
lecture2-allfiles.zip       unix-timeline.png
lecture2-slides.pdf
```

In [10]:
```
1 !git clone https://github.com/COMP2221/example-repository.git
```

```
Cloning into 'example-repository'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 13 (delta 0), reused 7 (delta 0), pack-reused 0
Receiving objects: 100% (13/13), done.
```

In [11]:
```
1 !ls
```

```
durhamlogo.png              lecture2-slides.pdf
example-repository          qr-poll.png
for-quiz                    rise.css
```


Git over the command line

git add, rm and commit

- git add: add new files
- git rm: remove files from git
    - --cached: keeps local copy
- git commit: commit your current changes


Git over the command line

git add, rm and commit

- git add: add new files
- git rm: remove files from git
    - --cached: keeps local copy
- git commit: commit your current changes

In [14]:

```
!mkdir src
!mv helloworld.c src
!ls
```

```
Makefile  README.md power.c   power.h   src
```


Git over the command line

git add, rm and commit

- git add: add new files
- git rm: remove files from git
    - --cached: keeps local copy
- git commit: commit your current changes

In [14]:

```
!mkdir src
!mv helloworld.c src
!ls
```

```
Makefile  README.md power.c   power.h   src
```

In [15]:

```
!git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    helloworld.c

Untracked files:
```

Git over the command line

git add, rm and commit

- git add: add new files
- git rm: remove files from git
    - --cached: keeps local copy
- git commit: commit your current changes

In [14]:
```
!mkdir src
!mv helloworld.c src
!ls
```

```
Makefile  README.md power.c   power.h   src
```

In [15]:
```
!git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    helloworld.c

Untracked files:
```


Git over the command line

git add, rm and commit

- git add: add new files
- git rm: remove files from git
    - --cached: keeps local copy
- git commit: commit your current changes

In [14]:
```
!mkdir src
!mv helloworld.c src
!ls
```

```
Makefile  README.md power.c   power.h   src
```

In [15]:
```
!git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    helloworld.c

Untracked files:
```


Git over the command line

git add, rm and commit

- git add: add new files
- git rm: remove files from git
    - --cached: keeps local copy
- git commit: commit your current changes

In [14]:
```
!mkdir src
!mv helloworld.c src
!ls
```

```
Makefile  README.md power.c   power.h   src
```

In [15]:
```
!git status
```

```
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        deleted:    helloworld.c

Untracked files:
```


Git over the command line

Git pull/push

- Pull: Get changes made to the repository
- Push: Add the changes you made to the repository


Git over the command line

Git pull/push

- Pull: Get changes made to the repository
- Push: Add the changes you made to the repository

In [ ]:

```
1 !git push
```

```
Username for 'https://github.com':
```


What is a commit hash?

- Everything is checksummed before it is stored and is then referred to by that checksum
- This detects changes to the contents of any file or directory
- Git stores everything in its database not by file name but by the hash value of its contents

Thus:

- Every commit has a corresponding hash that can be used to refer to it


What is a commit hash?

- Currently: Git uses a SHA-1 hash.
- A SHA-1 hash looks like this: 24b9da6552252987aa493b52f8696cd6d3b00373
- You will learn more about checksums in Networks and Systems
- This might change: https://git-scm.com/docs/hash-function-transition/
- Since it is not secure "enough": https://shattered.io/


Github classroom

- In the practical session:
  - set up your github classroom account
  - you will receive an invite link to an introductory "assignment" in the practical
  - This will contain more resources to learn how to use git
- Coursework:
  - You should be using git for all your big projects, including the coursework.
  - if you have any trouble setting your github classroom account please get in touch!


Summary

- UNIX
- Basic UNIX commands
- Git fundamentals