Operating Systems

6CM503

Computer GE 645 with OS Multix and its creators around

01 Assessment Brief 2025/26

Linux Command Line Interface (CLI): Shell Script Design

Dmitry A. Zaitsev

Module Leader

- Dr. Dmitry A. Zaitsev
- Email: d.zaitsev@derby.ac.uk, Phone: 01332592678
- Office: MS229. Recommended days for individual communication: Monday, Wednesday.
- Website: https://www.derby.ac.uk/staff/dmitry-zaitsev

Key dates and details

Assessment Detail	Assessment Information
Assessment Type:	Individual Report, Program Code
Assessment Weighting:	25%
Word count/Length:	5,000 words / 12 pages
Learning Outcomes:	1, 2
Submission Method:	Blackboard Assignment
Submission Date:	12:00 Noon UK time, 27/10/2025
Provisional Feedback Release Date:	12:00 Noon UK time, 10/11/2025

Description of the assessment

User interface is one of the basic interfaces of OS, together with interfaces with running processes and computer hardware. As a standard de facto, a general-purpose modern OS provides two kinds of user interface: Graphical User Interface (GUI) and Command Line Interface (CLI). GUI and CLI are mutually complementary. Besides, CLI is laconic and flexible; sequences of commands are composed into pipes and scripts that brings functionality of a programming language. Scripts play significant role in configuring OS, especially during booting and shutdown processes, as well as in configuring and installing application software.

The goal of the assessment is to master command line interface of Linux and Shell programming language to compose scripts, which play significant role in tuning of Linux and solving application tasks. Brief and powerful CLI allows us to administer Linux efficiently even via low-speed emergency remote connections.

With this assessment students master skills of composing systems scripts in shell (bash) language and they learn how to do it by and by, mastering command line language at firs and composing a pipe at second. Thus, these sequential advances facilitate better student experience. At first, they compose a sequence of commands to implement required actions to learn bash commands for: working with files, processes, pipes and filters; the task was adopted and extended from DEC authorized courses in Unix. At second, students compose a single but not elementary pipe, represented as a small script. At third, they compose a systems script for monitoring OS resources. For scripts, they have 12 variants assigned individually within the explicit list of students. Finally, the history of required command execution, and two scripts are submitted along with the report.

Basic information: Lectures 1-5.

Relationship to Programme Assessment Strategy

The first task of the module Coursework four portfolio tasks.

Attributes and Skills

The attributes and skills are listed in the following table.

	Skills	Links to useful resources
☒	Critical thinking	Analysis of historical and contemporary OSs.
☒	Communication	Working within common Linux server environment.
☒	Collaboration	Partial teamwork on the same task variants.
☒	Creative problem solving	Script, utility, and kernel loadable module design
☒	Self-direction & planning	Planning efforts and activities to satisfy task submission deadlines.
☒	Numeracy, statistics & financial literacy	Top X lists and statistics on most widespread OSs for specific application domains, awareness of financial aspects with OSs' comparative cost and reliability
☒	Digital	Students not only use but create digital technologies
☒	Resilience	Students learn how to avoid stress inflicted by with necessity to satisfy deadlines and recover after busy times
☒	Adaptability	Adaptation to using rather novel, for students, type of OS compared to officially supported by UoD MS Windows
☒	Leadership & future thinking	Developing kernel loadable modules, students think about OS of future

Assessment Content

Part I. CLI training

Find and run required command and select appropriate options

Workflow:

1. Files and directories
2. Protection of files
3. Processes
4. Filters: sort and grep
5. Save the command history together with outputs, add comments
6. Represent the file system tree transformation

CLI training sequences: Appendix 1.1.

Part 2. Compose a pipe

Compose a pipe that extends functionality of Linux as an additional command with the specified name and function, and test the pipe work.

Workflow:

1. Draft a sequence of required commands
2. Find out required options
3. Test each command separately
4. Incremental design adding commands to the pipe
5. Compose and test final pipe
6. Save the pipe in a file, add parameters

Variants of task: Appendix 2.

Part 3. Design a script

Compose a program in Shell (bash) language (script) according to a variant and test the script work.

Workflow:

1. Draft a sequence of required commands
2. Find out required options
3. Test each command separately
4. Save the script in a file, add parameters
5. Incremental design adding commands to the script
6. Add control flow to the script: branching and loops
7. Compose and test final script

Variants of task: Appendix 3.

Content of report: Appendix 4.

Assessment Rubric

Criteria	Excellent (70-100%)	Very good (60-69%)	Good (50-59%)	Satisfactory (40-49%)	Unsatisfactory (<40%)
CLI training, 20%	Successful implementation of all required actions	Successful implementation of all required actions with minor errors	Successful implementation of basic actions	Implementation of basic actions with some errors	Basic actions have not been implemented
Compose a pipe, 30%	Successful implementation of a given pipe	Successful implementation of a given pipe with minor imperfections	Successful implementation of basic functions of a given pipe	Implementation of basic functions of a given pipe with some errors	Basic functions of a given pipe have not been implemented
Design a script, 30%	Successful implementation of a given script	Successful implementation of a given script with minor imperfections	Successful implementation of basic functions of a given script	Implementation of basic functions of a given script with some errors	Basic functions of a given script have not been implemented
Report writing, 20%	The report is exceptionally well-structured, clearly written, and professionally presented.	The report is well-structured and clearly written with minor errors.	The report is complete but may lack clarity or detail in some sections.	Report is poorly organized or lacks clarity. Analysis is weak or incomplete.	Report is disorganized, difficult to follow, and lacks key elements.

Assessment AI-assistance

In this assessment AI-assistance is permitted in the following ways:

- Installation troubleshooting
- Check of obtained results

In this assessment AI-assistance should not be used for:

- Code composition
- Writing report

It is YOUR RESPONSIBILITY to check ALL information generated by generative AI tools. Any misuse of generative AI tools could be considered ethical academic misconduct (as per Academic Regulations, Academic Misconduct, Section J2). If in doubt, please consult your module leader.

Anonymous Marking

You must submit your work using your **student number** to identify yourself, not your name. You must not use your name in the text of the work at any point. When you submit your work in Turnitin you must submit your student number within the assignment document and in the *Submission title* field in Turnitin. Guidance is available showing how to do this.

Assessment Regulations

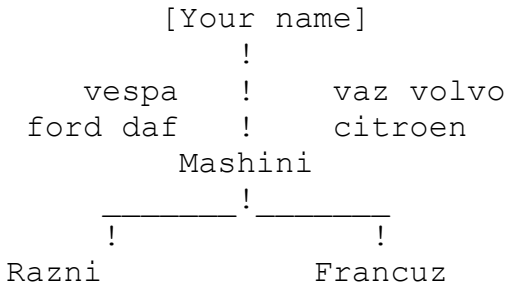
The University's regulations, policies and procedures for students define the framework within which teaching and assessment are conducted. Please make sure you are familiar with these regulations, policies and procedures.

Appendices

Appendix 1. Linux CLI training sequences for bash

A. Files and directories

1. Create the following structure of directories and files. Give Your name to the top directory. This directory will be your home directory.



2. Stay in Your home directory.
3. Move *ford* to directory *Mashini*.
4. Move all files, which names begin from "v" to directory *Razni*.
5. Move *vaz* to directory *Mashini*.
6. Stay in directory *Mashini*.
7. Copy *daf* to this directory.
8. Stay in directory *Raznie*.
9. Move *vespa* to directory *Francuz* and rename it into *vespa2*.
10. Create three files with arbitrary names.
11. Go to directory *Mashini*.
12. Copy two files from *Raznie* to *Francuz* using only one command.
13. Stay in Your home directory.
14. Show on the screen all levels under Your home directory.

B. Protection of files

1. Stay in Your home directory and execute the command

ls -al > LIST

which creates a file *LIST* showing initial protection of filers.

2. Let files *.profile* and *fruits* and directory *practice* are situated in Your home directory. They will be used further.

3. Change protection of file *.profile* that no other category of users except you can read the file content. Check the result.

4. Change protection of file fruits that only You or members of Your group can read file and write to it but all other users cannot do it.

5. Check which group files of Your directory belongs to. Check which group you belong to. Change using a command the group of a file. Check in file */etc/group* who belongs and who does not belong to Your group.

6. Cancel all access rights for all categories of users for directory practice. Execute command ls - al practice. Cancel Your own x-right. Execute command ls - al practice.

C. Processes

1. Determine who works in system and what processes are started by each user. For each process determine its state and priority. Find processes unattached to no one terminal.

2. Try to delete processes started by other users.

3. Create a command *dummy* which incessantly prints into standard output stream its name. Start the command. Interrupt its running and delete the process.

4. Start process *dummy* in background mode. Redirect its output into a file. Find the corresponding process in the processes list. Determine its state. Delete it.

5. Do the same operations decreasing the process priority using command *nice*. Change priority of running process using command *renice*.

6. Start a few copies of the process *dymmy* in background mode; use redirection of output into a dummy device. Find started processes in the processes list. Exit from the system. Enter the system. Check the processes list.

7. Do the same operations starting processes in uninterrupted mode *nohup*. Compare process states after repeated entering into the system. Delete started processes.

8. Enter commands of starting three dummy processes in various instances of time (for example, after 3, 5, and 7 minutes). Watch starting the processes then delete them. Do processes start after a user exits the system?

D. Filters: sort

1. Create a file named *family* that contains surnames, names and birth dates of a few persons. Each line has the following form:

smith john 3/1/50

Separate name and birth date with a tabulation (<tab>).

2. Use command *sort* to sort the file in alphabetic order.

3. Use an option (key) of command *sort* to sort the file in reverse order.

4. Create another file named *work* which contains surnames, names and birth dates of a few colleagues (group mates) having the same format as file family. Include Your personal data into the file.

5. Sort both files simultaneously by command *sort*. Pay your attention that your name is printed twice. Use an option (key) of command *sort* to print your name only once.

6. Create sorted files from files *work* and *family*. Use the key "–o" or redirection arrow ">". If You use ">" please choose a new file name for sorted file.

7. Try using the key "-m" of command sort to sort source files work and family. Then try to sort with the ket "-m" already sorted files. Pay attention that this key does not sort files but merges sorted files.

8. Sort Your files by the alphabetic order of name.

9. If a few persons have the same surname the program do not sort them automatically according to names. To overcome it, insert the key "-2" which enlarges sorting field till the column number 3.

10. Sort files in the order of the birth month. Try it with the key "+2". Look how it has been done for people born in October, November, December. Try doing it with the key "+2n".

E. Filters: grep

1. Create a file named price.veg with the following data

Lettuce	.89
tomatoes	0.89
broccoli	.79
cauliflower	0.89
parsely	2.14
avocado	1.19
carrots	0.69
ctltry	0.59
string-beans	1.29
onions	0.29
asparagus	1.39
corn	0.85
cabbage	0.99
lima-beans	1.29
etllow-beans	1.17
mushrooms	1.09
peppers	1.39
potatoes	0.39
artichokes	1.59
black-beans	1.89
blacheyed-beans	1.49

2. Sort the file twice: once in alphabetic order and twice according to prices. Write results into two new files.

3. Separate all beans in a file.

4. Find vegetables which names start with letter "s".

5. Find vegetables which names start with a vowel.

6. Find vegetables which price ends with digit "9".

7. Find vegetables which price ends with digits "09", "39", or "79".

8. Find vegetables which price does not end with digit "9".

9. Find vegetables which names do not start with letters from “a” to “m”.

10. Find vegetables which price ends with digit “9” but does not end with “39”, “59”, “79”, and “89”.

Directions:

- Use *bash*, better in a multi-user alpha-numeric screen mode.
- Print command history with *history*.
- Use your initials as a prefix for names of all created objects – file/directory, process, records of files, etc.
- For editing files use editor *vi* (see brief description in Appendix).

Appendix 2. Variants of task

Choose the task variant number according to the file of variants

1) lx – list of executable files of a specified directory.
2) pu – list of processes started by a specified user.
3) nx – number of executable files in a specified directory.
4) npu – number of processes started by a specified user.
5) np – number of processes running a specified file.
6) mp – list of users who started a specified (or greater) number of processes.
7) kn – delete all processes running a specified command.
8) bp – print information on a specified number of processes having the greatest time of running on processor.
9) bf – print information on a specified number of files having the gratest size.
10) kp – delete all stopped processes of a specified user.
11) ml – print information on a specified number of files having the greatest number of links.
12) ll – list of users who are owners of files in a specified directory.

Directions:

- Choose the task variant number according to the file of variants
- At first, execute required commands manually one-by-one.
- Then compose selected commands into a pipe.
- Remember the pipe in a file.
- Give to the file permission to execute it.
- Use *cut* to cut vertical columns
- Use *grep*, *head*, *tail* to manipulate rows.

Basic items:

1. Beginning/finishing work with system. Users, groups.
2. Files and directories. Protection of files (directories).
3. Process control.
4. Editing text.
5. Redirection of input/output. Pipes, filters.

Basic commands of Unix:

- ✓ Users and groups: *login, exit, who, newgrp, man*.
- ✓ Files and directories: *ls, cat, cp, mv, rm, pwd, cd, mkdir, rmdir, find (file, ln, diff, comm, cmp)*.
- ✓ Protection: *chmod, chown, chgrp, umask*.
- ✓ Editing files: *ed, vi*.
- ✓ Processes: *ps, &, kill, nice, at, nohup, renice*.
- ✓ Filters: *wc, sort, grep, sed, cut, paste, pr (awk)*.

Appendix 3. Variants for script composition

Variants of task:

1. Track users who started more than specified number of processes.
2. Track changing the number of file links in the specified sub-tree of directories.
3. Track users who started more than specified number of processes from one terminal.
4. Track changing access rights in the specified sub-tree of directories.
5. Track creating new files/directories in the specified sub-tree of directories.
6. Track changing free disk space for a specified disk.
7. Track changing access rights in in the specified sub-tree of directories.
8. Track login/logout of users into the system from specified terminal printing the time of user work.
9. Track deletion of files/directories in the specified sub-tree of directories.
10. Track changing processes priorities in specified range.
11. Track the list of terminals attached to processes with specified state.
12. Track processes which used the amount of processor time exceeding the specified value.

Directions:

- Choose the task variant number according to the file of variants
- Implement tracking via periodic (after 1-5 minutes) repeating of certain actions and comparing the current state with the previous state saved in an auxiliary temporary file.
- For periodic start use either command sleep or at.
- Create auxiliary temporary files in directory /tmp, provide unique names of temporary files (using built-in variable $$).
- Provide correct completion of the process (cleaning temporary files etc) on its deletion via processing of corresponding signals.
- The source of additional information – man-pages of shell (bash) and commands.

Basic items:

1. Structure of *shell* command line, interpretation of meta-symbols, patterns.

2. Variables of *shell*, built-in variables.
3. Arguments and options of commands; return codes; substitution of results of a command execution.
4. Control of operations sequences: branching and lops. Checking conditions.
5. Subroutines and signals processing in shell scripts.

Basic constructs of Shell language:

Meta-symbols:

> >> < <<s <<\s <<'s' |
. * ? [abc] [d-f]
; (command) {commands} & command1&&command2 command1||command2
'string' "string" `command`

Variables and parameters:

variable=value read variable echo $variable
$variable ${variable}
${variable-value} ${variable=value}
${variable?string} ${variable+value}
$1 $2 etc
shift
set argument1 argument2 etc

Appendix 4. Content of report

1. CLI training
   1.1. List of required commands (history of bash)
   1.2. Brief description of used commands
   1.3. Screen images of directory structure, processes, used files
2. Pipe
   2.1. How to use the pipe
   2.2. Pipe description
   2.3. Brief description of used commands
   2.4. Results of the pipe testing (no less than 3 tests)
3. Script
   3.1. How to use the script, working files specification
   3.2. Script description supplied with graphical schemata
   3.3. Brief description of used commands
   3.4. Results of the script testing (no less than 3 tests)

The report should be supplied with an archive containing (in textual files ready to run) a list of executed commands, pipe, and script supplied with comments on how to use them.

○