

Modelling, Uncertainty and Data for Engineers (MUDE)

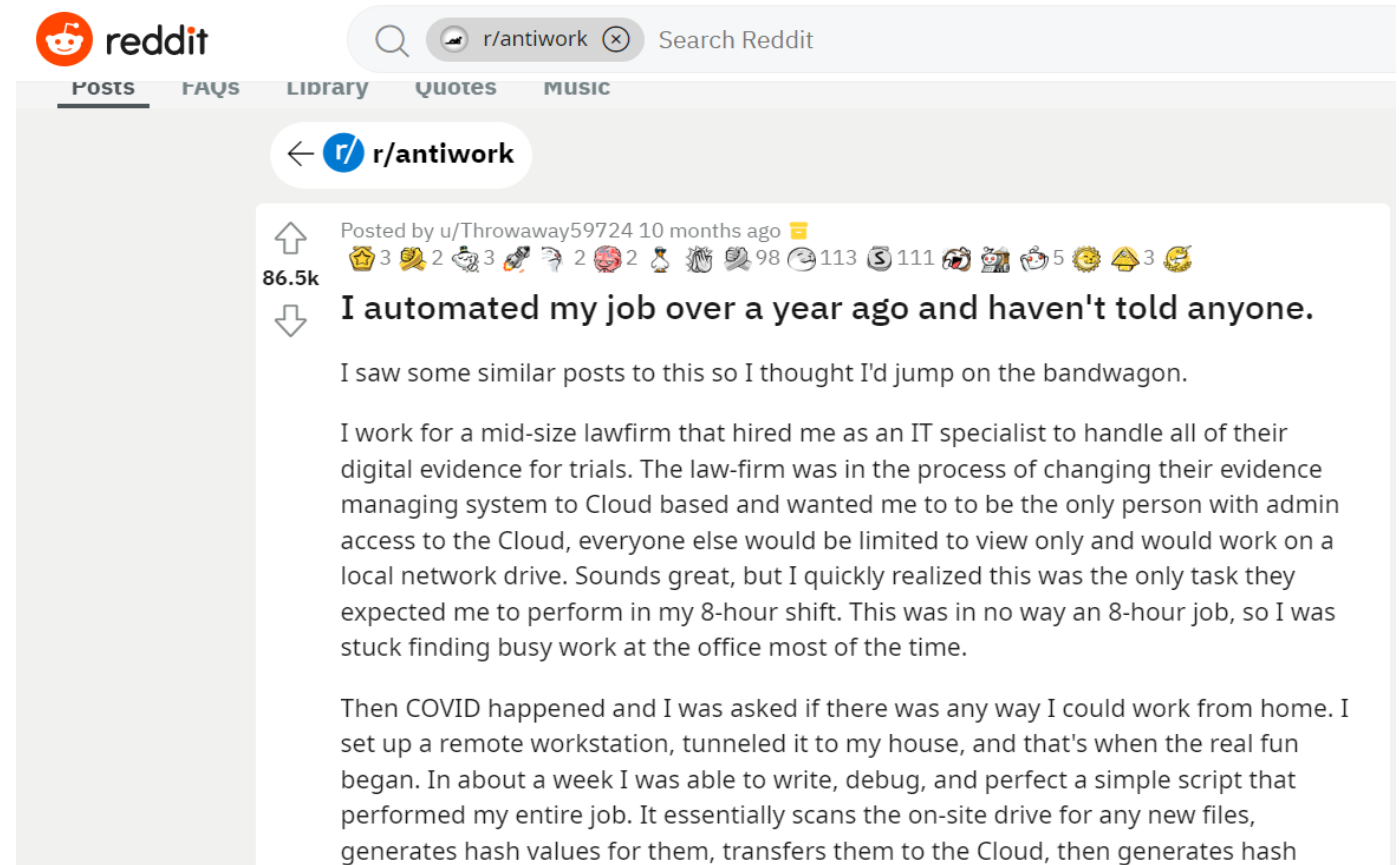
Basic Linux shell commands

Denis Voskov, Ilshat Saifullin

Shell introduction

- Shell is ...
 - command-line interface
 - scripting language
- Most popular is Bash
- No clicks as in GUI. Just write the commands and execute them
- Perfect for repetitive tasks
- Combine scripts and organize your workflow

Example of automation



https://www.reddit.com/r/antiwork/comments/s2igq9/i_automated_my_job_over_a_year_ago_and_havent/?sort=top

Shell command syntax

- General syntax

COMMAND OPTIONS ARGUMENTS

- Getting help

- COMMAND --help
- man COMMAND
- online help <http://man.he.net>

Example

- ls
- ls --help
- ls -l *.txt

```
lcur1922@r11n24:~$ ls --help
Usage: ls [OPTION]... [FILE]...
List information about the FILES (the current directory by default).
Sort entries alphabetically if none of -cftuvSUX nor --sort is specified.

Mandatory arguments to long options are mandatory for short options too.
-a, --all                        do not ignore entries starting with .
-A, --almost-all               do not list implied . and ..
--author                        with -l, print the author of each file
-b, --escape                    print C-style escapes for nongraphic characters
--block-size=SIZE              with -l, scale sizes by SIZE when printing them;
                               e.g., '--block-size=M'; see SIZE format below
-B, --ignore-backups            do not list implied entries ending with ~
-c                              with -lt: sort by, and show, ctime (time of last
                               modification of file status information);
                               with -l: show ctime and sort by name;
                               otherwise: sort by ctime, newest first
-C                              list entries by columns
--color[=WHEN]                 colorize the output; WHEN can be 'always' (default
                               if omitted), 'auto', or 'never'; more info below
-d, --directory                list directories themselves, not their contents
-D, --dired                    generate output designed for Emacs' dired mode
-f                              do not sort, enable -aU, disable -ls --color
-F, --classify                 append indicator (one of */=>@|) to entries
--file-type                    likewise, except do not append '*'
--format=WORD                  across -x, commas -m, horizontal -x, long -l,
                               single-column -l, verbose -l, vertical -C
--full-time                     like -l --time-style=full-iso
-g                              like -l, but do not list owner
--group-directories-first
```

Shell keys

Key	Description
Up	Previous command
Ctrl+R then type	Search in command history
Ctrl+C	Interrupt the command
Tab	Autocomplete the command or filename

Shell commands: files and directories

Commandls	Description	Examples		
ls	List current directory	ls	ls ./logs/1?.txt	ls -l
pwd	Print current directory path	pwd		
cd <i>DIR</i>	Change directory	cd ~	cd ..	cd /usr/include
mkdir <i>DIR</i>	Make a new directory	mkdir data		
cp <i>SRC DEST</i>	Copy a file or directory	cp 1.txt 2.txt	cp *.txt ./logs	cp -r dir_1 dir_2
mv <i>SRC DEST</i>	Move/rename a file or directory	mv old.txt new.txt		mv new.txt data
rm <i>FILE</i>	Remove the file or directory. Note: removed files cannot be	rm new.txt rm log*.txt rm -r data		

Shell commands: files and directories

Command	Description	Examples
touch <i>FILE</i>	Create an empty file with name <i>FILE</i>	touch my_file.txt
which <i>FILE</i>	Print the path to executable	which python
tar -czf <i>ARCH INPUT</i>	Create an archive with name <i>ARCH</i>	tar czf data.tar.gz data data_2
tar -xzf <i>ARCH</i>	Unpack the archive	tar xzf data.tar.gz
chmod <i>MODE FILE</i>	Change mode for the <i>FILE</i> MODE: u/g/o/a +/- r/w/x	chmod a+x *.sh
find	Finds files by name pattern, type, ...	find mydir/'pattern'

Shell commands: tools

Command	Description	Examples
echo <i>TEXT</i>	Display a line of text	echo "Hello!"
cat <i>FILE_1 FILE_2 ...</i>	Concatenate files and print to std output	cat cities.txt cat 1.txt 2.txt
head -n 2 <i>FILE</i>	Output the first # lines of file(s)	head -n 10 cities.txt
tail -n 2 <i>FILE</i>	Output the last # lines of file(s)	tail -n 10 *.txt
wc <i>FILE</i>	Print line, word, byte counts for file(s)	wc -l cities.txt
grep <i>TEXT FILE</i>	Print lines of file that match patterns	grep Den Haag cities.txt

Shell commands: pipes

Command	Description	Examples
COMMAND > FILE	Redirect output to the FILE	cat 1.txt 2.txt > 12.txt
COMMAND >> FILE	Redirect output to the FILE in append mode	cat 1.txt >> 12.txt cat 2.txt >> 12.txt
COMMAND < FILE	Use FILE as input to COMMAND	tail -n 10 *.txt
CMD1 CMD2	Use output of the command CMD1 as input to the command CMD2	cat cities.txt grep Rotterdam

Shell: Variables

- Shell variables are treated as strings
- Variables are assigned using "=" symbol
- Add "\$" before the name of the variable to use it
- The PATH variable defines the shell's search path

Example:

```
a=1
b=5
c=a+b
c=$((a + b))
```

Command	Description
echo \$PATH	Show the PATH variable value
s="Hello" echo \$s	Set a variable and print the value.
export s="Hello"	Set as an environment variable . Other processes started from this shell will use it.

Shell: loops

```
for f in *.txt
do
    echo $f
done
```

```
for f in *.txt; do echo $f; done
```

```
for f in *.txt
do
    tail -n 10 $f >> all.txt
done
```

```
for entity in temp pressure
do
    for value in 5 10 15
    do
        echo $entity $value
    done
done
```

```
for i in $(seq 5 10); do echo $i; done
```

Shell: conditions

```
if CONDITION
then
COMMANDS_1
else
COMMANDS_2
fi
```

```
# spaces in [ ] are mandatory
if [ $i == 5 ]
then
echo "I is 5"
else
echo "I is not 5"
fi
```

Operator	Description
&&	logical and
	logical or
-gt	> (greater)
-ge	>= (greater or equal)

```
# spaces in [ ] are mandatory
if [ $i -gt 5 ]
then
echo "I greater than 5"
fi
```

Shell: system information

Command	Description
<code>cat /proc/cpuinfo</code>	Processor info
<code>uname -a</code>	OS info
<code>lsb_release -a</code>	OS version
<code>df -h</code>	Free space on disk
<code>free</code>	Amount of free and used memory (RAM)
<code>top -c</code>	Processes (-c to show command arguments)

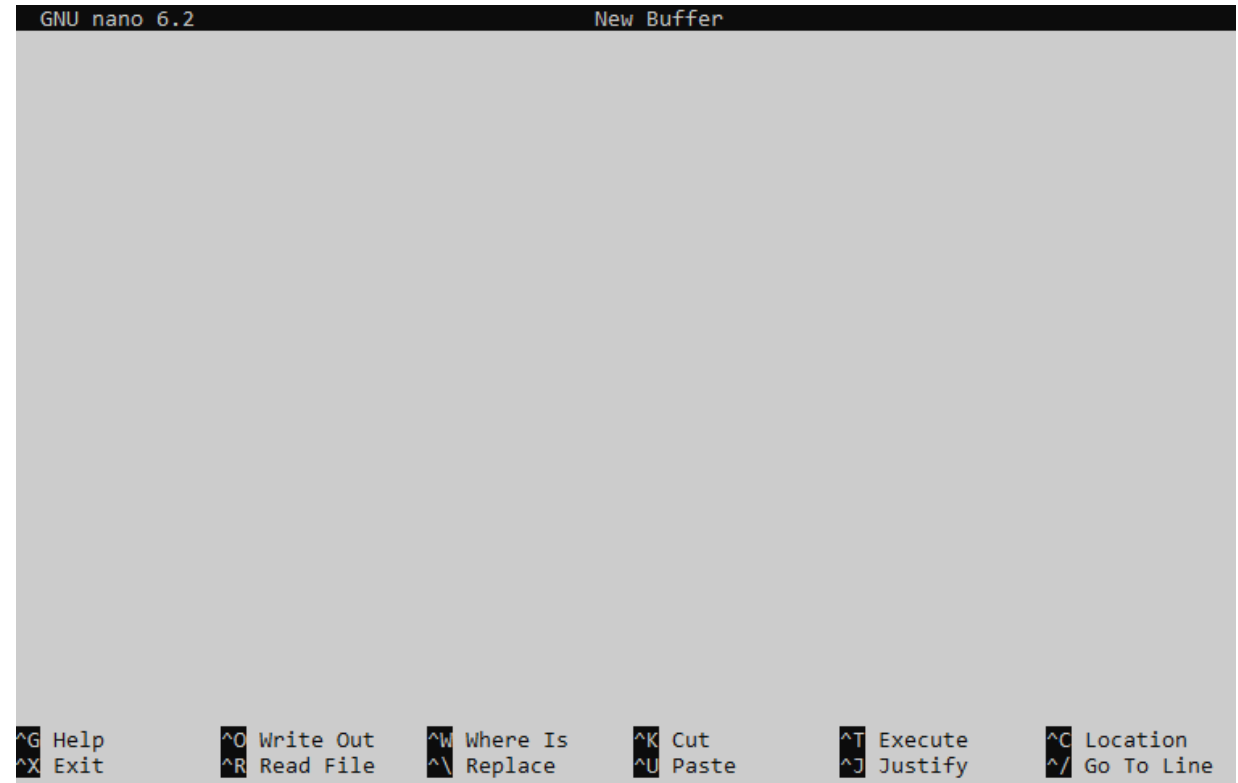
Shell: processes and signals

Command	Description
<code>./my_script.sh</code>	use <code>./</code> to run executable file from current directory (if not in <code>\$PATH</code>)
<code>time ./my_script.sh</code>	Run and measure the resource usage

```
touch my_script.sh
chmod a+x my_script.sh
./my_script.sh
time ./my_script.sh
```

Text Editor (nano)

Command	Description
nano my.txt	Open the editor
Ctrl+G	Show Help
Ctrl+O	Save the file
Ctrl+X	Exit



```
GNU nano 6.2          New Buffer

^G Help  ^O Write Out  ^W Where Is  ^K Cut      ^T Execute  ^C Location
^X Exit  ^R Read File  ^\ Replace   ^U Paste    ^J Justify  ^_ Go To Line
```

Bash script file

1. **#!/bin/bash** at the first line
(interpreter)

```
GNU nano 6.2
#!/bin/bash
echo "Hello"
```

2. Using cmd arguments: **\$#**, **\$1**

```
echo "Script $0 is running with $# arguments."
echo "The first argument is $1"
```

3. Commenting: **#**

```
user@TUD258093:~$ ./my_script.sh 50
Script ./my_script.sh is running with 1 arguments.
The first argument is 50
```

Exercise 1: count files and run script

1. Create a text file with your bash commands
2. Depending on the argument, print the number of files in directory
 - a) If no arguments: in directory `Week_2_2` (recursively)
 - b) Argument is `sub_dir`: in directory `Week_2_2 /sub_dir`
3. Check that the sum of results of two calls with arguments `well_logs` and `well_tests` is equal to result of call without arguments. Use variables.

```
user@TUD258093:~$ ./files_count.sh
Script ./files_count.sh is running with 0 arguments.
55
user@TUD258093:~$ ./files_count.sh well_logs
Script ./files_count.sh is running with 1 arguments.
The first argument is well_logs
30
user@TUD258093:~$ ./files_count.sh well_tests
Script ./files_count.sh is running with 1 arguments.
The first argument is well_tests
25
```


Exercise 2: search, print and collect the data

1. Depending on the argument ***well_name***, print the temperature from file Week_2_2 /well_tests/***well_name***_welltest.txt
2. Collect all the filenames and well temperatures to one file

```
/mnt/c/Users/isaifullin/soft~/well_tests/W10_welltest.txt
@@@ Well-test report, powered by DARTS (https://darts.citg.
@@@          Special edition for MUDE 2022-2023
Well name: W10
Perforation interval: 1998 to 2074
Fluid in place: brine
Initial pressure:      201.035
Temperature:          344.440
```

```
Temperature:          339.194
4shell_excercise/well_tests/W20_welltest.txt
Temperature:          346.606
4shell_excercise/well_tests/W21_welltest.txt
Temperature:          349.855
4shell_excercise/well_tests/W22_welltest.txt
Temperature:          338.977
4shell_excercise/well_tests/W23_welltest.txt
Temperature:          352.395
4shell_excercise/well_tests/W24_welltest.txt
Temperature:          344.258
```

Exercise 3: use archive and process files

1. Create an archive with well_test files for the wells with names: W3, W5, W15, W17.
Files are in the directory:
well_tests/**well_name**_welltest.txt
2. Create a new directory **my_data** and copy all the files from Week_2_2 directory to it. Remove the files for the wells W1, W2, .. , W10 in **my_data**