Universität Göttingen
Institute for Computer Science / GWDG
Kevin Lüdemann

Linux Crash Course Exercise 1 / 2024-10-07
HPC System Administration / WiSe 2024/25
55 Minutes Total

Exercise Introduction

Before starting the exercise, make sure you have the slide deck for the Linux Shell Crash Course ready and you have a Bash shell under Linux before you. You can use the GWDG machines or any other Linux system with an up to date Bash shell, such as a local virtual machine.

The goal of these exercises is to make you familiar with the Bash shell, so feel free to play around with it, test things out and either ask for help or search for help online. This crash course only introduces a subset of the commands available in Bash and there is yet another myriad of tools that can be installed from the internet. Focus on understanding the Bash shell, its commands and shortcuts so you can productively work with it rather than perfectly completing all exercises.

When copying out commands, depending on the PDF reader you are using, spaces might be lost such that the command does not work. Check with the command in the PDF and add missing spaces.

Contents

Task 1: Follow Along (20 min)

This exercise walks you through the commands shown in the slides. You do not have to perfectly follow all steps, experiment with the commands if it helps you to get a better understanding of them.
Try using `TAB` to auto-complete commands and file/directory names.
Use `ARROW-UP/DOWN` to cycle through your command history and reuse or edit past commands if it means typing less.

Folder Navigation

`sleep 60`	Sleep for 60 secs.
`CTRL`+`c`	Interrupt the command.
`cd`	Switch to your home directory.
`pwd`	
`ls`	
`ls -a`	
`ls -l`	
`ls -la`	
`mkdir shell-ex`	Create folder shell-ex.

`cd shell-ex`	
`mkdir "delete me"`	
`rmdir delete me`	Observe that it tries to delete *delete* and *me*.
`rmdir "delete me"`	
`ls -a ..`	List parent directory.

Help

`mkdir --help`	
`man mkdir`	See different ways of getting help.
`q`	Pager can be quit using `q`.
`man --help`	
`man -h`	
`man man`	Open the first page of the manual for man.
`whatis man`	See what pages are available for command man.
`man 7 man`	Open page 7 of the manual for man.

Permissions

`cd ~/shell-ex`	
`mkdir perm-ex`	
`cd perm-ex`	
`touch file.txt`	
`mkdir folder`	
`ls -la`	
`chmod a-r file.txt`	Remove read permission.
`chmod a-r folder`	
`ls -la`	
`ls folder`	Try to read the folder.
`cat file.txt`	These should both fail.
`touch folder/newfile.txt`	Writing new files is still okay.
`chmod a= file.txt`	Remove all permissions.
`chmod a= folder`	
`rm file.txt`	Try to delete file.txt.
`rmdir folder`	Try to delete folder. This should fail.
`chmod u+r folder`	Add read permissions back.
`ls folder`	
`touch folder/file2.txt`	Try to create another file. This should fail.
`chmod u+w folder`	Add back write permission.
`touch folder/file2.txt`	Try again to create another file. This should still fail.
`chmod u+x folder`	You need execution permission on a folder to create files.
`touch folder/file2.txt`	

Nano

`cd ~/shell-ex`	
`mkdir nano-ex`	
`cd nano-ex`	
`nano`	Create and start editing a buffer.
Write some text and one very long line.	
`CTRL`+`o` Name it file.txt	Save your file.
`CTRL`+`x`	Exit nano.
`cat file.txt`	
`nano file.txt`	
Make a change.	
`CTRL`+`x`	Try to exit without saving.
Answer the prompt with `n` or `y` and `ENTER`.	

Environmental Variables

```
echo $HOME
echo $PWD
echo $PATH
echo -e ${PATH//:/:\\n}
printenv                                    See all variables, depending on host, this might be a lot.
export HELLO=hello
echo $HELLO
export HELLO="$HELLO world"                 Append to a variable.
echo "$HELLO"
echo '$HELLO'
unset HELLO
echo $HELLO
nano ~/.bash_profile
```

Add the line `[[ -f ~/.bashrc ]] && .  ~/.bashrc`
Save and exit nano.

```
nano ~./bashrc
```

Add the line `export HELLO=hi`
Add the line `alias ll='ls -la'`
Save and exit nano.   Feel free to add more aliases that seem useful.

```
source ~/.bashrc
ll                                          Try out the new alias.
echo $HELLO
```

File and Folder operations

```
cd ~/shell-ex
mkdir operations-ex
cd operations-ex
mkdir folder
touch file
mv file folder                              Move file into the folder.
mv folder/file file.txt                     Move it back out and rename it from file to file.txt.
cp file.txt folder
cp folder folder2                           This should fail.
cp -R folder folder2
```

Reading and Searching

```
cd ~/shell-ex
mkdir read-search-ex
cd read-search-ex
man man > man.txt                           Use a redirection to create a file with the output from
                                            man.
```

`head man.txt` View the first 10 lines of man.txt.
`tail man.txt` View the last 10 lines of man.txt.

```
head -n 20 man.txt                          View the first 20 lines of man.txt.
grep manual man.txt                         Show all lines containing manual in man.txt.
grep -c manual man.txt                      Count the number of occurrences of manual.
grep -wc manual man.txt                     Count the number of occurrences of manual as a whole
                                            word.

cp man.txt man2.txt
nano man2.txt
```

Make some changes, write text, delete some lines.

```
diff man.txt man2.txt                       See your changes.
```

Command	Description
`wc man.txt`	Print line, word and byte counts for man.txt.

Processes

Command	Description
`top`	Get an overview of current resource usage.
`htop`	Get a better overview of current resource usage.
`ps`	Get a list of all your current processes.
`ps -ef`	Get a list of all currently running processes.
`sleep 60 &`	Run sleep in a background job.
`kill PID`	Enter the process id returned by the previous command.
`export HELLO2=hi2 && echo $HELLO2 && unset HELLO2 && echo $HELLO2`	Chain commands using &&.

Redirection

Command	Description
`mkdir ~/shell-ex/redirect-ex && cd ~/shell-ex/redirect-ex`	
`ps -ef > p.txt`	Write output of command into a file.
`echo $HOME >> p.txt`	Append output to file.
`tail p.txt`	
`ps -ef	grep ssh`	Pipe the output of one command into another.
`ps -ef	grep -wc root`	Count the number of processes involving root.
`ps -ef	grep root	sort -nk 2	head`	Get the first 10 processes involving root by pid.
`ps -ef	head -1; ps -ef	sort -r -nk 3	head -15`	Get the 15 processes with the highest CPU consumption.
`!!`	Use the previous command again.
`echo "alias bycpu='!!'" >> ~/.bashrc`	Turn previous command into an alias called `bycpu`.
`source ~/.bashrc`	
`bycpu`	Test our your new alias.

Bash History

Command	Description
`history`	View your command history.
`history	grep ps`	Find all commands including ps.
`history	less`	Open history in a pager.
`!NUMBER`	Insert a number from history to repeat that command.
`!ps`	Expands to the last used command starting with ps.
`!?grep`	Expands to the last used command containing grep.

wget & curl

Command	Description
`mkdir ~/shell-ex/wget-curl-ex && cd !#:1`	!#:1 refers to the second word of the current command.
`wget gwdg.de`	See that it downloads the html document into index.html.
`wget -O gwdg.html gwdg.de`	Now its saved to gwdg.html instead.
`curl gwdg.de`	See that it prints to the shell instead.
	The request needs to be redirected and curl did not follow it automatically.
`curl -L gwdg.de`	The -L flag follows the redirect.
`curl -Lo gwdg2.html gwdg.de`	Combine the flags.
`lynx gwdg.de -dump	less`	Lynx is a terminal browser, it can also be used directly.
`tar -cvzf gwdg.html.tar.gz gwdg.html`	Create an archive from gwdg.html.
`rm gwdg.html`	Remove the original file.
`tar -xvzf gwdg.html.tar.gz`	Extract it again.
`zip gwdg.html.zip gwdg.html`	Use zip instead of tar & gzip.
`rm gwdg.html`	
`unzip gwdg.html.zip`	Unpack again.
`ls -alh`	See the file sizes of the archives and the regular files.

Further Reading

- Advanced Programming in the UNIX Environment 3rd edition by R. Stevens and S. Rago

Task 1: For the Advanced (20 min)

This is a more difficult **optional** task which can be done instead of Task 1

Work on these task. You do not need to complete them all or in that order, focus on those that interest you.

- Find out how to use curly brackets { } to not type common sub-strings in arguments twice (e.g. `mv file{.txt,.md} folder`)
- Customize your `PS1` variable and save it to your .bashrc.
- Find examples that use `PS2` , `PS3` , `PS4` , `PS5`
- Send yourself an email using Bash (sendmail, mail and mailx are available)
- Find out how to use `trap` command
- Find out how to use the `awk` scripting language within Bash

Task 2: Bash Scripting Basics (5 min)

`cd`	Switch to home directoy.
`mkdir script-ex && cd !#:1`	Create a folder for the exercises.
`nano first.sh`	
Write `#!/usr/bin/bash` as the first line.	
Write `echo "Hello World!"` as the second line.	
Save and exit nano.	
`chmod u+x first.sh`	Add execution permission.
`./first.sh`	Run it.

Further Reading

- `https://linuxhint.com/30_bash_script_examples/`

Task 3: Bash Scripting data collection (30 min)

The idea of this task is to use a bash script for collecting data over time stored in a system file. We will focus on the CPU frequency for now and write it for a single core into a file. This file will contain two columns. The first column is a time stamp like this `20240319_094726` . The second column is the current frequency of core0. The steps of the task are outlined below but you will need to get more knowledge about the commands, so RTFM.

- First you will need to find out about the date command. Run `man date`
- Save the time stamp into a variable
- Find the file that contains the frequency of core0
- Use `cat` to store the frequency in a variable

- Echo the content of the variables into a file with a pipe

- Put all the commands into a bash script

- Write a loop that appends these values every second to the same file

Further Reading

- https://linuxhint.com/30_bash_script_examples/

Task 3: Add more cores, free memory and temperatures (30 min)

This is a more difficult **optional** task which can be done instead of Task 3

Tracking one core is usually not useful in a many core system. Therefore, you can update the script you have written before and include all cores.

Additionally, the amount of free memory (RAM) is an interesting information to have, especially for debugging memory leaks. Find the file containing the amount of free memory and update the bash script to store this value in an additional column.

Not completely relevant in an HPC setting but for home computers and laptops, use the tool called `sensors` to read out the CPU temperature. If you are running this on a local machine or VM you may need to install the `lm-sensors` package. Use the `grep` command and string manipulation to extract the relevant text. This part is not possible on the SCC since this package is not installed.

Once you are done, think about showing the content of the file graphically. Use a plotting program like `gnuplot` or plotting libraries for python or any other language.

You can lookup commands and how to use them via `man` or on the internet.

Hints

- Use https://www.shellcheck.net/ to check whether your syntax is valid.
- `echo -e` Enables backslash escapes such as `\t` for tabs.
- `echo -e "Date: ‘date‘"` This will execute the command within '.
- Use `cut` to reduce the output of commands, for example, `w | cut -d ' ' -f1` gives a list of all users. `echo -e ` w | -d ''-f1 ` ` to ignore new lines.
- `echo -e "CPU Usage:\t" ` cat /proc/stat | awk '/cpu/{printf("%.2f%\n"),` `($2+$4)*100/($2+$4+$5)}'| awk '{print $0}'| head -1 ` ` Gives current CPU usage.
- You can use colored outputs like this
  `RED='\033[0;31m'` `NC='\033[0m'` `echo -e "Default ${RED}colored text${NC}Blank text"`
  Colors are `'\033[0;30m'` to `'\033[0;37m'` and `'\033[1;30m'` to `'\033[1;37m'`
- You can use functions to organize your code:
  `function_name(){`
  `  # Function code`
  `}` You can call the functions like this `$(function_name)`