

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 (30 min)	1
Task 1: For the Advanced (30 min)	5
Task 2: Bash Scripting Basics (5 min)	5
Task 2: System Overview Script (5 min)	5
Task 3: Git (15 min)	6

Task 1: Follow Along (30 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

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

```
cd shell-ex
mkdir "delete me"
rmdir delete me
rmdir "delete me"
ls -a ..
```

Observe that it tries to delete *delete* and *me*.

List parent directory.

Help

```
mkdir --help
man mkdir
q
man --help
man -h
man man
whatis man
man 7 man
```

See different ways of getting help.

Pager can be quit using `q`.

Open the first page of the manual for man.

See what pages are available for command 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
chmod a-r folder
ls -la
ls folder
cat file.txt
touch folder/newfile.txt
chmod a= file.txt
chmod a= folder
rm file.txt
rmdir folder
chmod u+r folder
ls folder
touch folder/file2.txt
chmod u+w folder
touch folder/file2.txt
chmod u+x folder

touch folder/file2.txt
```

Remove read permission.

Try to read the folder.

These should both fail.

Writing new files is still okay.

Remove all permissions.

Try to delete file.txt.

Try to delete folder. This should fail.

Add read permissions back.

Try to create another file. This should fail.

Add back write permission.

Try again to create another file. This should still fail.

You need execution permission on a folder to create files.

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

`CTRL` + `x`

Save your file.

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
export HELLO=hello
echo $HELLO
export HELLO="$HELLO world"
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.
source ~/.bashrc
ll
echo $HELLO
```

See all variables, depending on host, this might be a lot.

Append to a variable.

Feel free to add more aliases that seem useful.

Try out the new alias.

File and Folder operations

```
cd ~/shell-ex
mkdir operations-ex
cd operations-ex
mkdir folder
touch file
mv file folder
mv folder/file file.txt
cp file.txt folder
cp folder folder2
cp -R folder folder2
```

Move file into the folder.

Move it back out and rename it from file to file.txt.

This should fail.

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
grep manual man.txt
grep -c manual man.txt
grep -wc manual man.txt
```

View the first 20 lines of man.txt.

Show all lines containing manual in man.txt.

Count the number of occurrences of manual.

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.

<code>wc man.txt</code>	Print line, word and byte counts for man.txt.
Processes	
<code>top</code>	Get an overview of current resource usage.
<code>htop</code>	Get a better overview of current resource usage.
<code>ps</code>	Get a list of all your current processes.
<code>ps -ef</code>	Get a list of all currently running processes.
<code>sleep 60 &</code>	Run sleep in a background job.
<code>kill PID</code>	Enter the process id returned by the previous command.
<code>export HELLO2=hi2 && echo \$HELLO2 && unset HELLO2 && echo \$HELLO2</code>	Chain commands using &&.
Redirection	
<code>mkdir ~/shell-ex/redirect-ex && cd ~/shell-ex/redirect-ex</code>	
<code>ps -ef > p.txt</code>	Write output of command into a file.
<code>echo \$HOME >> p.txt</code>	Append output to file.
<code>tail p.txt</code>	
<code>ps -ef grep ssh</code>	Pipe the output of one command into another.
<code>ps -ef grep -wc root</code>	Count the number of processes involving root.
<code>ps -ef grep root sort -nk 2 head</code>	Get the first 10 processes involving root by pid.
<code>ps -ef head -1; ps -ef sort -r -nk 3 head -15</code>	Get the 15 processes with the highest CPU consumption.
<code>!!</code>	Use the previous command again.
<code>echo "alias bycpu='!!'" >> ~/.bashrc</code>	Turn previous command into an alias called <code>bycpu</code> .
<code>source ~/.bashrc</code>	
<code>bycpu</code>	Test our your new alias.
Bash History	
<code>history</code>	View your command history.
<code>history grep ps</code>	Find all commands including ps.
<code>history less</code>	Open history in a pager.
<code>!NUMBER</code>	Insert a number from history to repeat that command.
<code>!ps</code>	Expands to the last used command starting with ps.
<code>!?grep</code>	Expands to the last used command containing grep.
wget & curl	
<code>mkdir ~/shell-ex/wget-curl-ex && cd !#:1</code>	<code>!#:1</code> refers to the second word of the current command.
<code>wget gwdg.de</code>	See that it downloads the html document into index.html.
<code>wget -O gwdg.html gwdg.de</code>	Now its saved to gwdg.html instead.
<code>curl gwdg.de</code>	See that it prints to the shell instead.
	The request needs to be redirected and curl did not follow it automatically.
<code>curl -L gwdg.de</code>	The -L flag follows the redirect.
<code>curl -Lo gwdg2.html gwdg.de</code>	Combine the flags.
<code>lynx gwdg.de -dump less</code>	Lynx is a terminal browser, it can also be used directly.
<code>tar -cvzf gwdg.html.tar.gz gwdg.html</code>	Create an archive from gwdg.html.
<code>rm gwdg.html</code>	Remove the original file.
<code>tar -xvzf gwdg.html.tar.gz</code>	Extract it again.
<code>zip gwdg.html.zip gwdg.html</code>	Use zip instead of tar & gzip.
<code>rm gwdg.html</code>	
<code>unzip gwdg.html.zip</code>	Unpack again.
<code>ls -alh</code>	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 (30 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)

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

Further Reading

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

Task 2: System Overview Script (5 min)

This is a more difficult **optional** task which can be done instead of Task 2

Create a bash script that gives an overview of the current system and its resource usage.

Add the script as an alias to your `.bashrc`.

Incorporate the outputs of the following commands in some form in your script:

- `hostname`
- `uptime`
- `uname -r`
- `arch`
- `w`

- `free`
- `hostnamectl`
- `lscpu`
- `hostname -I`

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)`

Task 3: Git (15 min)

Run the following commands and observe what they do. Feel free to test around.

You can find help for any git command using `git COMMAND --help`

Setup

```
mkdir -p $HOME/git-ex && cd $HOME/git-exercise
git --version
git config --global user.name "NAME"           Set your name.
git config --global user.email "EMAIL"         Set your email.
git config --global core.editor "nano"         Set nano as editor for commit messages.
git init --initial-branch=main
git status
```

Committing

```
touch README
git status
git add .
git status
git commit -m "Initial Commit"
nano README                                     Write a few words into the file and close nano.
git status
```

```
git diff README
git add README
git commit
```

See your changes, close with `q`.

Write a commit message using nano, save and close.

Reverting changes

```
rm README
git status
git reset --hard HEAD
```

Undo the delete by reverting to the last commit this also undoes any other changes you made.

```
git status
ls
rm README
git commit -a -m "Deleted README"
git status
ls
```

See that the README file is back.

Use `-a` flag to commit a staged changes.

```
git revert TAB + TAB
```

Confirm that README was deleted and the change was committed.

This shows a list of your recent commits.

Type the first two characters of the id of your last commit and press `TAB` and `ENTER`.

Write a commit message for your reverted commit, save and close nano.

Remote repository

```
git remote add origin "https://gitlab-ce.gwdg.de/USERNAME/git-exercise.git"
git push --set-upstream origin main
```

Make sure you can login to <https://gitlab.gwdg.de>, <https://gitlab-ce.gwdg.de> or <https://github.com>.

Replace the domain and username accordingly.

This will query your credentials if you do not have them stored already

and create a remote repository.

The visibility of the repository is private by default so only you and the teammates you have explicitly invited have access.

Visit <https://gitlab-ce.gwdg.de/USERNAME/git-exercise> to view your new project.

Delete the local copy of the repository.

switch back to your home directory.

```
rm -rf /git-exercise
cd
git clone https://gitlab-ce.gwdg.de/USERNAME/git-exercise.git
```

Download the repository from remote.

```
cd git-exercise
```

Make a change to README on the web and commit it.

```
git pull
git log
```

See the change you made on the web.

gitignore

```
touch credentials
git status
nano .gitignore
git status
```

See that the credentials file can be staged.

Write **credentials** and save.

See that only **.gitignore** can be staged and credentials is ignored.

```
git add credentials
```

Git has many more features, one of them, which is commonly used, is branching.

Further Reading

- Missing Semester; Version Control (Git): <https://missing.csail.mit.edu/2020/version-control/>
- Learn Git branching: <https://learngitbranching.js.org/>