HPS
https://hps.vi4io.org

Kevin Lüdemann

Linux Shell Crash Course

Surviving the Terminal

Learning Objectives

- Connect to GWDG machines via SSH and access the command line interface
- Navigate the operating system on Linux using the Bash shell
- Edit files using Nano text editor
- Solve routine tasks by formulating commands and combining existing programs

Preface

- Focus on most important commands
- Additional content for advanced users
- Use this slide deck as lookup during course
- Available for download on course page:
  `https://hps.vi4io.org/teaching/summer_term_2026/pchpc`
- ← Red box marks a command that you want to remember
- Other commands are nice to know
- Presentation accompanied by exercises
- Support room: `https://meet.gwdg.de/b/jul-ffv-ljs-7u5`

What is a Shell?

- A shell is a command line interpreter
- It takes commands entered via the keyboard to start programs
- **Bash** is the most widespread shell
- A **terminal** is an input/output environment for shells
- The mouse can still be used to select text for copy and paste
- The shell is only an interface through which other programs are started
- A shell can only show textual output

Open a shell:

- **Windows:** WIN + r, type `powershell` and press enter
- **MacOS:** Search for **Terminal** and open it

Table of contents

SSH Client

Windows 10/11:

- Use the **Powershell**
- Confirm that it works with `ssh -V`
- Alternatives: Mobaxterm, putty

MacOS/Linux:

- Search for **Terminal** and open it
- Check your ssh version `ssh -V`

Logging in

- Generate a key: ssh-keygen -t ed25519
- Upload key to `id.academiccloud.de`
- ssh -i <your-ssh-key> uxxxxx@glogin.hpc.gwdg.de
- ssh -i <your-ssh-key> uxxxxx@glogin-gpu.hpc.gwdg.de
- Documentation: `docs.hpc.gwdg.de/start_here/connecting`

Basic Command Syntax

- Common syntax for commands is `COMMAND <-OPTIONS> <ARGUMENTS>`
- A command might take 0 or more options prefixed with a `-` and separated by spaces (long options use `--`)
- A command might take 0 or more arguments separated by spaces
- Arguments can be subcommands that also accept options
- Arguments including spaces must be put in quotes `"my argument"`
- `" "` allow for variable expansion, `''` do not

Syntax Example

- First command `echo`
- It prints whatever you type after it
- Try `echo hello world`
- It accepts the option `-e` to enable escape commands
- Try `echo -e "hello\nworld"` (try without `-e`)

Filesystem Hierarchy

- In Linux, everything is a file
- Directories are separated via `/` (Same for Mac, Windows has `\`)
- For example, `/path/to/my/folder` (directory and folder are used interchangeably)
- `/` is the root directory
- `.` indicates the current folder `./my/folder`
- A path can be absolute (starting with `/`)
  or relative to the current directory (starting with `.`)
- Parent of current directory is `..`

Folder Navigation

- `pwd` Print current directory
- `ls` List files and folders in current directory
- `ls -a` Also list hidden files and folders (start with . marks as hidden)
- `ls -la` List all files and folders in long table format
- `ls -a DIR` List all files and folders in target directory
- `cd DIR` Change directory to target directory
- `cd ~` Change to HOME directory
- `cd ..` Change to parent folder
  - `~` Refers to your HOME folder
  - `.` Refers to the current folder
  - `..` Refers to parent of current folder
- A path including spaces `cd "path/with spaces/"` needs to be put in quotes

Create, Copy, Move, Delete

- `touch FILE` Update modification time of file or create empty file
- `rm -i FILE` Delete file with confirmation, confirm with `y`
- `mkdir DIR` Create directory
- `rmdir DIR` Delete directory
- `rm -rf DIR` Delete everything in folder (sub-folders, files, ...)
  use with **great care**, there is **no undo**
- `cp SRC DEST` Copy a file from source to destination
- `cp -R SRC DEST` Copy folders including sub-folders
- `mv SRC DEST` Move a file or folder, also functions as rename

Read and Search Files

- `cat FILE` Print file content to shell
- `less FILE` Show file content with pager
- `find PATH -name '*.txt'` Find all txt files in path
- `locate NAME` Find files containing NAME in their filename
- `grep PATTERN FILE` Search for pattern in file
- `grep -R PATTERN PATH` Search for pattern in all files in path
- `head FILE` Show first 10 lines of file
- `tail FILE` Show last 10 lines of file
- `diff FILE1 FILE2` Compare files and list differences

Shell Shortcuts Basics

- TAB Auto-complete file/directory names and commands
- TAB + TAB Show all possibilities
- CTRL + c Abort current running process
- ARROW UP/DOWN Cycle through command history
- `clear` Clear screen
- `exit` Close current shell session

Getting help with a command

- `COMMAND --help`, `COMMAND -h` or `COMMAND help` commonly shows usage options
- `man COMMAND` Opens the manual for a command
  - Mouse wheel for scrolling
  - d / w For scrolling down/up
  - Mouse wheel sometimes does not work via SSH
  - q For quitting the manual
  - Try `man man`
- `whatis COMMAND` See what pages are available
- `man SECTION COMMAND` Open a specific page for a command
- Search for documentation and guides on the internet

Nano Basic Usage

- **Nano** is a text editor for the terminal
  - Relatively easy to use
  - Alternatives: **emacs**, **vi**, ...
  - Use your preferred editor
- `nano FILE` To start editing, if file does not exist, its created
- Navigate with ARROW-keys and type to edit
- CTRL + o To save as...
- CTRL + s To save (HPC machines have old nano, use CTRL + o instead)
- CTRL + x To exit

Nano Shortcuts 1/2

- ESC Can be used instead of ALT
- CTRL + w Open search
- ALT + w Continue search
- CTRL + w, CTRL + R Open search and replace
- CTRL + c Cancel command
- ALT + a Set mark for selection
- ALT + 6 Copy selected text (area between mark and cursor) to clipboard
- CTRL + k Cut current line or selected text to clipboard
- CTRL + u Paste clipboard at cursor

Nano Shortcuts 2/2

- ALT + u/e Undo/Redo
- CTRL + a/e Jump to line start/end
- CTRL + y/v Scroll page up/down
- CTRL + g Open help window
- CTRL + o Save as..
- CTRL + c Show cursor position
- CTRL + 7 Jump to line number
- ALT + o Enable/Disable conversion of tabs to spaces

Environmental Variables

- Values can be stored in environmental variables
- Some are used for configurations
- `echo $HOME` To see the value of HOME
- `echo -e ${PATH//:/:\\n}` To get a nice output for PATH
- `printenv` or `set` to see all current env vars
- `export NAME=Value` Set variable, no spaces before or after =
- `unset NAME` Unset variable
- Env vars are bound to your session and do not persist after session ends

Persistent settings

- When you login into a Bash shell, it reads `.bash_profile`
- When you open another Bash shell without login, it reads `.bashrc`
- `nano .bash_profile` Open bash profile and make it load .bashrc
- Add this line to it `[[ -f ~/.bashrc ]] && . ~/.bashrc` and save
- `nano .bashrc` To start editing
- Add `export HELLO=hi`
- `alias` Can be used to set command aliases
- Add `alias ll='ls -la'` and save
- `source .bashrc` To load the changes now

Custom Prompt

- By setting the env var `PS1` you can customize your prompt
- Try `export PS1='[\t] \u@\h:\w$'`
- `\t` Gives the current time
- `\u` Gives your username
- `\h` Gives the hostname
- `\w` Gives the current folder
- Search for **bash ps1 generator** on the internet

Redirect Command Outputs

- `COMMAND > FILE` Redirects the output of command into file
- `>` Creates or overwrites file, `»` creates or appends file
- `|` A pipe that forwards inputs from one command into another
- `ps aux | grep PATTERN` Filter the output of a command using grep
- `COMMAND | sort -u` Sort and filter unique lines in output
- Only the output of the last command is shown in the shell

Bash History

- `history` List all previous commands
- `history -c` Clear history (in case you entered your password)
- `history | grep PATTERN` Look for a command you used before
- `!N` Expands to line n of your bash history
- `!!` Expands to previous command
- `!TEXT` Expands to last command starting with text
- `!?TEXT` Expands to last command containing text
- `!#:N` Expands to nth argument of current command, can be used like this:
  - `mkdir NEW_DIR && cd !#:1` to create and switch to new dir

File and Folder Permissions

- Files and folders each belong to a user (owner) and a group
- Read, write and execute permission can be set for owner, group and others
- `ls -l` shows these permissions

d	rwx	- - -	- - -	2	linuxuser	UMIN	1	Mar 31 14:42	test
-	rw-	- - -	- - -	1	linuxuser	UMIN	16533	Mar 31 14:41	test.txt
type	user perm	group perm	other perm	# of links	owner	group	size	last modified	name

- Type **d** means directory, **-** means file
- Permission **-** means its not set, **r**, **w**, **x** means read, write or execute permission set

Modifying Permission

- `chmod` Command for changing permission
- `chmod (u|g|o|a)(+|-|=)(r||w||x|| ) TARGET`
- `chmod a+r test.txt` Gives everyone read permission
- `chmod g= test.txt` Removes all permission for group
- `chmod u+x test` Allows execution of test
- `chmod -R g+rwX test-dir` Makes test-dir and files and folders in it group readable and writable, **-R** flag makes it recursive

Changing ownership

- `chown NEW_OWNER TARGET` Change the ownership of target
- `chgrp NEW_GROUP TARGET` Change the group of target
- The admin or super-user on Linux systems is called root
- `sudo COMMAND` (super-user do) Execute command as admin
- `whoami` Show own username
- `who` Show logged in users
- `w` More information active users

Processes

- `top` or `htop` Show current resource usage by processes
  Use htop over top, close with `q` or `CTRL` + `c`
- `ps` List all processes on current shell session
- `ps -u USER` List all processes by a specific user, try `ps -u root`
- `ps aux` or `ps -ef` List all processes by all users
- `kill PID` Stop process with process id
- `COMMAND1 && COMMAND2` Lets you chain multiple commands
  this will execute COMMAND1 and then COMMAND2
  but only if COMMAND1 succeeded

Jobs

- `COMMAND &` Let the command execute as a background job
- CTRL + z Stop and make the running command a background job
- `jobs` List your background jobs
- Jobs are bound to your shell session, all jobs are killed when you close your shell
- `bg %JOB_NUM` Start a stopped background job
- `fg %JOB_NUM` Move a job into the foreground
- `disown %JOB_NUM` Disown a job from your shell, keeps it running after closing shell

Shell Scripting

- Bash commands can be used to program shell scripts
- Written in plain text and saved as `.sh` files
- Must have as first line `#!/usr/bin/bash`
- You can use loops, conditions and so on
  like a regular programming language
- Make it executable if it isn't `chmod u+x script.sh`
- Run a script using `./script.sh`
- First inspect a script `less script.sh` or `nano script.sh`
  before running it
- Commonly used to start jobs on supercomputers

Postface

- Linux networking was not covered
- Git for Windows comes with the Git Bash shell, which contains most Bash commands `https://gitforwindows.org/`
- Terminal under MacOS uses either Bash or Zsh by default, check your shell with `echo $SHELL` and the version of Bash with `bash --version`
- Find more Bash tricks `https://github.com/dylanaraps/pure-bash-bible`
- Guide on Bash `https://learnxinyminutes.com/docs/bash`
- Detailed command lookup `https://explainshell.com/`

Conclusion

- The shell is really powerful
- It does not restrict you to the options of graphical programs
- You can combine outputs from multiple programs
- Automate your boring workflows
- With experience you can become very productive
- Further reading for the interested:
  `https://lwn.net/Articles/343828/`

  `https://arcan-fe.com/2022/04/02/the-day-of-a-new-command-line-interface-shell/`