Bachelorprogrammierkurs 2024
Bash Basics

Aaron van der Graaf (TU Dortmund)
09-04-2024

Was ist Bash?

- Bash = Bourne-again **Shell**
- Meist genutzte **Shell** unter Unix-Systemen
  - Einzige (falls kein GUI) oder ergänzende Benutzerschnittstelle
- [< ***username*** > @ < ***node*** > < ***dir*** >]$ < ***command*** >
- Tabulator Taste ermöglich Autovervollständigung → Sehr mächtig

```
(base) [avdgraaf@libertas Useful_Scripts]$ ls -lh
total 126K
-rwxr-xr-x. 1 avdgraaf e4 1.8K Feb 28 16:17 Check_for_Nans.py
-rw-r--r--. 1 avdgraaf e4  965 Feb 21 14:16 Check_Number_Events_Iterate.py
-rw-r--r--. 1 avdgraaf e4 1.1K Feb 21 14:16 Check_Number_Events.py
-rw-r--r--. 1 avdgraaf e4 7.2K Feb 21 14:16 Copy_Filespathes_to_Txt.py
-rwxr-xr-x. 1 avdgraaf e4 1.8K Feb 21 14:16 Filter_txt_2.0.py
-rw-r--r--. 1 avdgraaf e4 1.2K Feb 22 16:24 Fix_Nans.py
-rwxr-xr-x. 1 avdgraaf e4 5.3K Feb 21 14:16 Get_relative_Sys.py
```

Erinnerung: ssh

- ssh (**S**ecure **Sh**ell) zum sicheren Einloggen auf die Infrastruktur
- Nur neptun direkt von außer zugänglich → Siehe IT-Onboarding
- Nützlich um direkt auf die Workstation zuconnecten → **Proxy Jump**
  - ssh –J neptun < workstation >
  - Am besten direkt in der ssh config einbauen
- X11-Forwarding für grafische Anwendungen: ssh –X neptun
  - Langsam, besser alternative nutzen (Jupyterhub, MobaXterm, …)
  - Ebenfalls als Argument in die ssh config einbaubar

ssh config

- So in etwa sollte eure ssh config auch aussehen !
- Proxy Jump für meine Workstation (libertas)
- Hat jeder ein funktionierende ssh config?

```
Host neptun
  Hostname neptun.e4.physik.tu-dortmund.de
  User avdgraaf
  IdentityFile ~/.ssh/ssh_privat.txt
  ForwardX11Trusted yes
  ForwardX11 yes
  IdentitiesOnly yes
  ServerAliveInterval 15

Host libertas
  User avdgraaf
  Hostname libertas
  IdentityFile ~/.ssh/ssh_privat.txt
  IdentitiesOnly yes
  ForwardX11 yes
  ForwardX11Trusted yes
  ServerAliveInterval 15
  ProxyJump neptun
```

Navigation und Information

Cheat-Sheet

- Aktueller Order ist ./
- Ein Ordner darüber ist ../
- Home-Ordner ist ~/
- Vorheriger Order ist -/
- Ordnerinhalt anzeigen: ls < options > <dir>
  - Zum Beispiel: ls –lah
  - -l = List → Zeige Dateien in Listen Form an, eine per Reihe mit vielen Details
  - -a = All → Zeige alle Dateien an, auch versteckte
  - -h = human-readable → Zeige Dateigröße in passender Größe an sodass lesbar
  - Es gibt noch viele weiter hilfreiche Flags

Ein genauer Blick auf ls –lah

- Zeichenkette: Filetype+rwx (owner, group, world)
- Danach: Number of file links
- Owner, group, size (Größe nicht “korrekt” für Ordner → du –sh)
- Änderungsdatum für Dateien und Erstellungsdatum für Ordner

```
(base) [avdgraaf@libertas Useful_Scripts]$ ls -lh
total 127K
-rwxr-xr-x. 1 avdgraaf e4 1.8K Feb 28 16:17 Check_for_Nans.py
-rw-r--r--. 1 avdgraaf e4  965 Feb 21 14:16 Check_Number_Events_Iterate.py
-rw-r--r--. 1 avdgraaf e4 1.1K Feb 21 14:16 Check_Number_Events.py
-rw-r--r--. 1 avdgraaf e4 7.2K Feb 21 14:16 Copy_Filespathes_to_Txt.py
-rwxr-xr-x. 1 avdgraaf e4 1.8K Feb 21 14:16 Filter_txt_2.0.py
-rw-r--r--. 1 avdgraaf e4 1.2K Feb 22 16:24 Fix_Nans.py
-rwxr-xr-x. 1 avdgraaf e4 5.3K Feb 21 14:16 Get_relative_Sys.py
-rw-r--r--. 1 avdgraaf e4 1.3K Feb  6 12:46 Merge.sh
-rw-r--r--. 1 avdgraaf e4 1.3K Feb  6 13:38 Print.sh
-rw-r--r--. 1 avdgraaf e4  684 Sep 18  2023 Run_All_Sys_relativ_Diff.sh
drwxr-xr-x. 2 avdgraaf e4    2 Apr  8 10:00 Test
```

Order und Dateien anlegen und verändern

- **mkdir** <name>
  - Legt neuen Ordner an → *make directory*
  - Diverse Flags vorhanden, z.B. –p (parents) → Erstelle benötigte Unterordner
    - mkdir –p /some/new/directories/
- **cp** <path> <target>
  - Kopiert Dateien und Order → *copy*
  - Diverse Flags vorhanden, z.B. –r (recursive) → Benötigt fürs kopieren von Ordnern
    - cp –r Folder_211 ~/.
- **mv** <path> <target>
  - Verschieben oder Umbenennen von Dateien und Ordner → *move*
    - mv myfile.txt newname.txt
- **touch** myfile.txt
  - Legt neue (leere Datei an) → Keine Gefahr des Überschreibens, erneuert Timestamp falls existiert

Dateien Löschen

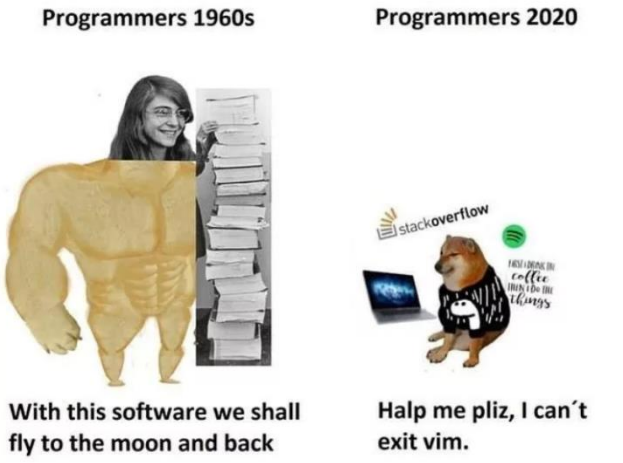
- **rm** <path>
  - Löschen von Dateien und Ordnern → *remove*
  - Diverse Flags vorhanden: z.B. -r (recursive) fürs löschen von Ordnern
- **Achtung: Dateirettung nicht in allen Fällen möglich!**
  - Wenn man sicher gehen will: Interactive mode mithilfe der -i Flag
- **Nicht empfehlenswert:** Verwendung der Flag -r (force)
  - Löscht Schreibgeschütze Dateien ohne Nachfrage
  - Kann jedoch hilfreich sein, z.B. beim löschen von Git Repos → Enthalten meist diverse schreibgeschützte Dateien → Sehr nervig ohne -f, da man X mal das löschen bestätigen muss

Wildcards

- Viele ähnliche Dateien entfernen/verschieben kann mühselig sein
- Abhilfe schaffen Wildcards → *
  - * steht für eine beliebige Zeichenkette
- Beispiele:
  - rm *.pdf → Löscht alle PDF-Dateien im aktuellen Ordner
  - rm Ordner/* → Löscht alle Dateien in dem Ordner (aber nicht den Ordner selbst)
  - cp Script*.py ~/. → Kopiert alle Python Dateien vom aktuellen in Order in dein Home Ordner die mit Script beginnen
- Hilfreiche Variante: ? steht für ein Zeichen
  - cp ?.txt ~/. → Kopiert alle txt Dateien mit einem Zeichen ins Home Verzeichnis

Texteditoren in der Konsole

- Grundsätzlich sollte man immer Alternativen verwenden wenn möglich, manchmal nicht möglich
- **Nano** → standardmäßiger Texteditor in Unix-Systemen
  - Anfangs sehr ungewohnt
  - Sehr Basic im Vergleich zu VS-Code
- Für den Anfang:
  - Text ändern
  - Mit STRG+X Speichern
- Andere besser Texteditoren vorhanden:
  - **vim**, …




```
  GNU nano 5.6.1                                   Test.py
import uproot
import numpy as np
import os

# Constants
PREFIX = "/ceph/groups/e4/groups/atlas_ana/data/data_aaron/Model_without_more_Vars/Ntuples_with_NN>
POSTFIX = ".root"

FILE_NAMES = [
"mc16a/410470", "mc16d/410470", "mc16e/410470", "mc16a/364100", "mc16a/364101", "mc16a/364102", "m>
]
BRANCH_NAME = "lep_PromptLeptonImprovedInput_RelCaloClusterE_1"

# lep_PromptLeptonImprovedInput_CandVertex_normDistToPriVtxLongitudinalBest_1
# lep_PromptLeptonImprovedInput_CandVertex_normDistToPriVtxLongitudinalBest_1
# lep_PromptLeptonImprovedInput_RelCaloClusterE_1
# lep_PromptLeptonImprovedInput_PtFrac_1

TREE_NAME = "nominal"

total_events = 0

                                   [ Read 54 lines ]
^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
```

SCP: Secure Copy

- Kopieren von Dateien mithilfe von ssh
  - scp -r ~/Cool_Folder libertas:~/.
  - Rekursives kopieren wieder möglich mit der -r Flag
  - Kopiert Ordner von aktuellen System mithilfe von ssh auf Zielsystem, hier libertas
  - Verwendeter Befehl benötigt funktionierende ssh config! Geht auch ohne → Nicht empfohlen
- Für Kopieren zwischen euren Laptop und Workstation muss der Befehl von eueren Laptop aus ausgeführt werden
  - Ansonsten nicht möglich, da ihr euch nicht von der Workstation aus auf euren Laptop connecten könnt.
- Alternativen möglich via VS-Code, mounten usw.

Übung: Erste Schritte im Terminal

- Freies Hands-On Tutorial: Probiert die folgenden Kommandos aus
- Kein Pflichtprogramm, nur Vorschläge
  - Navigiert durch die Directories (Home und ceph)
  - Untersucht Ordner und Dateien
  - Legt Ordnerstrukturen an
  - Erstellt und schreibt in Dateien
  - Kopiert ganze Order und Dateien
  - Löscht Dateien
  - Probiert Wildcards aus
  - Verwendet SCP → Beide Richtung, von Laptop auf Workstation und von Workstation auf Laptop
- Bei Unklarheiten: Immer fragen!

Was ist die .bashrc ?

- Eue erstes Bash-Script: nano ~/.bashrc
- Benutzerprofil für euere Bash → Einstellung
- Wird automatisch ausgeführt, sobald ihr eine interaktive Shell startet
- Üblicherweise genutzt um ein Alias zu definieren
- **Achtung**: condor-jobs haben keinen Zugriff auf .bashrc und somit keinen Zugriff auf euere Aliase, da sie keine interaktive Shell ist!

```
#Own Aliases
alias load_root="setupATLAS && lsetup 'root 6.28.04-x86_64-centos7-gcc12-opt'"
alias cd_fit="cd ~/fit/"
alias cd_fit_old="cd /ceph/groups/e4/users/avdgraaf/private/ana-topq-2021-14/fit"
```

Bash Scripting

- Files mit der Endung .sh sind Shell Scripte
- Enthält eine List von Commands die beim Ausführen des Scriptes von **einer Shell** ausgeführt werden → source Test.sh
- Praktisch für repetitive Aufgaben
- Shell Scripte sind meist recht einfach können jedoch auch beliebig kompliziert werden
- Verfügbare Keywords/Commands:
  - Shell Keywords: if, else, break, while, …
  - Shell Commands: cd, ls, echo, pwd, touch, …
  - Funktionen
  - Control flow: if…then..else, case, loops, etc.

➢ **Viel ist möglich**, nur wie?

Basics: Shell Scripting

- sh Scripte sollten mit dem **shebang** Konstrukt beginnen
  → Befiehlt die Bourne Again Shell zu nutzen
- All vorherigen Befehle (mkdir, mv, cp, …) können in den Scripten verwendet werden
- **<u>Neue Commands</u>:**
  - **echo**: Gibt den folgenden String aus
  - **Read**: Liest den gegebenen String in eine Variable ein
- **<u>Variablen:</u>**
  - Definition von Variablen mithilfe von **=**
  - Variablennamen dürfen nur **Buchstaben**, Zahlen und **Unterstriche** enthalten
  - Zugriff auf Variablen via **$**
  - **Achtung:** Leerzeichen und Tabs verändern den Code

```
#!/bin/sh
echo What is your Name?
read name
echo Hello $name!
```

```
(base) avdgraaf@SeJeY-PC:~$ source Test.sh
What is your Name?
Aaron
Hello Aaron!
```

```
#!/bin/sh
Var_1=100
Var_2=500
echo $Var_1 $Var_2
```

```
#!/bin/sh
Var_1 = 100
Var_2 = 500
echo $Var_1 $Var_2
```

```
(base) avdgraaf@SeJeY-PC:~$ source Test2.sh
100 500
```

Arrays & Basic Operations

- Arrays können per z.B. via **Array[0]=42** definiert werden
- Besser: **Array=(value1 … valueN)**
- Zugriff auf Array via:
  - **${Array[0]}** einzelnes Element
  - **${Array[*]}** für alle Elemente
- Addition, Subtraktion, Multiplikation und Division können direkt in Bash ausgeführt werden
  - **Einschränkung**: Nur für Integers
- Für Fließkommazahlen müssen externe Programme verwendet werden z.B. Python

```
#!bin/sh
PhD_at_E4=(Aaron Lukas Michael Donna)
echo ${PhD_at_E4[*]}
PhD_at_E4[0]=Donna
PhD_at_E4[3]=Aaron
echo ${PhD_at_E4[*]}
```

```
(base) avdgraaf@SeJeY-PC:~$ source Test3.sh
Aaron Lukas Michael Donna
Donna Lukas Michael Aaron
```

```
#!/bin/sh
x=42
y=10
z=$(($x * $y))
echo $z

a=1.1

ax=$(python -c "print($a*$x)")
echo $ax
```

```
(base) avdgraaf@SeJeY-PC:~$ source Test4.sh
420
46.2
```

If, Else und Loops

- If, Else und Loops können recht normal in Shell Scripten verwendet werden
- **Besonderheiten: fi, then, do, done**

```
#!/bin/sh

for var1 in 1 2 3
do
        for var2 in 0 5
        do
                if [ $var1 -eq 2 -a $var2 -eq 0 ]
                then
                        break 2
                else
                        echo "$var1 $var2"
                fi
        done
done
```

Command Line Arguments

- Arugmente können per $1 $2 … $n an die Scripte weiter gegeben werden
- Besonders Praktisch um Scripte adaptive zu machen, z.B: Path vorgeben zum Erstellen von Ordnerstrukturen oder ähnliches
- Weitere Command Line Arguments:
  - $0 Name des Scripts
  - $$ Werte der Argumente
  - $# Anzahl der Argumente

```
#!/bin/sh
echo $0
echo $1
echo $2
```

```
(base) avdgraaf@SeJeY-PC:~$ source Test6.sh 33 44
-bash
33
44
```

Scripte Ausführbar machen

- Andere Möglichkeit zum Ausführen von Scripten: **./myscript.sh**
- Unterschied: Script wird in einer neuen Shell ausgeführt, Output wird in der vorherigen Shell ausgegeben
  - Neue Shell = Nur .bashrc geladen → Vorher definierte Variablen sind nicht vorhanden!
- Um **Ausführbar** zu sein müssen die Rechte stimmen: u+x myscript.sh
- **Achtung:** Das **Shebang** muss im Script vorhanden sein, damit klar ist welche Shell verwendet werden soll. Deswegen immer **Shebang!**
- Alternative: **bash myscript.sh** → Führt Script in einer neuen Bash-Shell aus

Zusammenfassung Shell Scripte

- Sehr mächtig und praktische für viele Aufgabengebiete
- Schreibeweise häufig etwas kryptisch
- Leerzeichen sind der Feind jedes Shell Script Anfänger
- Fast alles ist möglich in Shell Scripten → **Stackoverflow**

Zusammenfassung Shell Scripte

- Sehr mächtig und praktische für viele Aufgabengebiete
- Schreibeweise häufig etwas kryptisch
- Leerzeichen sind der Feind jedes Shell Script Anfänger
- Fast alles ist möglich in Shell Scripten
  → ~~Stackoverflow~~ **Chat GPT**

Beispiel für ein Shell Script aus meinen Arbeitsalltag

```
#!/bin/bash

# Load python 3.6+ as it is needed -> load_sw is an alias, jsut load some proper cvmfs software package
load_sw

# Define the paths where the corresponding fit output files can be found
path_CB_Z="/eos/user/v/vandergr/Run3_2024_Validation_New/Merged/out/"
path_CB_Jpsi="/eos/user/v/vandergr/Run3_2024_Validation_New/Merged/out/"

#Define the paths where the plots shall be written to
out_path_CB="/eos/user/v/vandergr/WebEOS/Validation_2024_April/"
#out_path="/eos/user/v/vandergr/WebEOS/Rel22_Validation_2024/"

cd /afs/cern.ch/user/v/vandergr/private/MomentumValidation/source/MomentumValidation/scripts/

python MMC_full_plotting_refactored.py -r Z -t CB -f Eta -i $path_CB_Z -o $out_path_CB --directCB True --extension .png --which_run Run-3 --which_period e
python MMC_full_plotting_refactored.py -r Jpsi -t CB -f Eta -i $path_CB_Jpsi -o $out_path_CB --directCB True --extension .png --which_run Run-3 --which_period e
python MMC_full_plotting_refactored.py -r Z -t CB -f Pt -i $path_CB_Z -o $out_path_CB --directCB True --extension .png --which_run Run-3 --which_period e
python MMC_full_plotting_refactored.py -r Jpsi -t CB -f Pt -i $path_CB_Jpsi -o $out_path_CB --directCB True --extension .png --which_run Run-3 --which_period e


# Spread index file for WebEOS
cd /eos/user/v/vandergr/WebEOS
source Spread_index_file.sh
cd ~
```

Erweiterte Befehle

- **cat head tail**: Ausgeben des Inhalts einer Datei
  - **cat**: printe komplettes File
  - **tail**: letzte 10 lines
  - **head**: erste 10 Lines
  - -n: Anzahl der Lines
  - Noch viele weitere Flags möglich

- **grep**: Suche Lines in File mit bestimmten Wort
  - Zum Beispiel: **grep std::cout SickScript.cxx**
  - -i: Case-insensitive
  - -R: Recursive → Mehrere Dateien durchsuchen

Wer ist eingeloggt (w) ?

- Der command ***w*** zeigt alle Shells an, die gerade auf dieser Maschine laufen
- **TTY**: Shell-Name, **IDLE**: Zeit seit letzter Tastatureingabe
- JCPU, PCU: CPU-Zeit für Shell und What-Prozess (gerade laufend)
- Recht unzuverlässig (CPU-Zeit, Background-Prozesse)
  - **Usecase:** Vor Reboot (Anfragen) checken, ob jemand anderes gerade auf der Workstation rechnet, immer erst fragen!

```
(base) [avdgraaf@libertas Scripts]$ w
 15:12:43 up 117 days,  4:52,  4 users,  load average: 0.22, 0.36, 0.51
USER     TTY        LOGIN@   IDLE   JCPU   PCPU WHAT
bgocke   pts/0     08:42    6:30m  0.09s  0.06s screen -R fuenf
bgocke   pts/1     08:42    2days  1:38m  1:38m python3 -u -Wignore /cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase/x86_64/rucio-clients
avdgraaf pts/2     12:19    2:53m  0.01s  0.01s -bash
avdgraaf pts/5     15:01    0.00s  0.05s  0.00s w
```

tmux: persistente ssh-Verbindung

- ssh-Verbindungen basieren auf Datenaustausch
  - Probleme, wenn Internat ausfällt, Laptop zugeklappt wird, usw.
- Typische Fehlermeldung im Terminal: ***Broken Pipe***
- Sehr unangenehm, weil Prozess mit undefiniertem verhalten beendet (getötet) wird
- Abmildern mit ssh-config Einstellung (z.B. ServerAliveInterval → Siehe Slide 4)
- Bessere Lösung: **tmux**
- Mehr Infos dazu im Confluence

Terminal Hotkeys

- Folgende Hotkyes sind üblich und bekannt:
  - **STRG+C, STRG+V, STRG+V** (Copy, Cut, Paste)
- **STRG+D** schließt Terminal (= exit)
- Wichtiger: **STRG+c** zum Abbrechen eines Prozesses (kann sich noch Aufräumen)
- Falls das nicht geht: **STRG+z** zum Suspenden eines Prozesses (kein Aufräumen)
  - Prozess kann später wieder gestartet werden

Textumleitung

- Bash hat drei Kanäle, die Programmen zugewiesen werden:
  - stdin: Liest Eingabe von Tastatur ein
  - stdout: Standardausgabekanal
  - stderr: Standardfehlerkanal
- Umleiten in Dateien sinnvoll für Automatisierung oder viel Text
- Beispiel: ls ~/Desktop > verzeichnis.txt (>> zum Anhängen)
- **Achtung:** ggf. wir verzeichniss.txt überschrieben!
- Pipe-Operator nützlich: ***ls *.py | wc –l***
  - ***ls *.py*** zeigt alle Dateien mit der Endung *.py an. Die Liste an Dateien wird an ***wc*** weiter geben, dieser zählt mit ***-l*** die Anzahl der Zeilen und gibt somit die Anzahl an Python Dateien aus

Größe von Ordnern anzeigen

- Befehl: **du -sh ***
  - Größe aller Ordner anzeigen
  - -h für “human readable”
  - -s für Summarize (nicht rekursiv)

```
(base) [avdgraaf@libertas ~]$ du * -sh
497K    CSV
167M    Desktop
512     Documents
20M     Downloads
143M    Fake_SM_Tinkering
```

Zusammenfassung

- **Niemand** kann alle Befehle am Anfang auswendig
  - Man lernt die Begriff Stück für Stück
- Eine Liste der wichtigsten Shell- und Git-Befehle gesammelt auf Cheatsheet
  - Siehe Bachelorprogrammierkurs im Confluence oder eine anderes Cheatsheet → Siehe Slide 5

- Was **ihr** von dieser Session **mitnehmen solltet**:
  - Sehr **viel ist möglich** in Shell Scripten
  - Hauptanwendung: **Automatisierung** von **repetitiven Workflows**
  - Shell Scripte **selber schreiben** ist **doof** ☹
  - **ABER**: **ChatGPT** und Co. sind für sowas perfekt, so lange **Ihr** wisst **was geht** und **was nicht geht**

Übung: Advanced Commands ausprobieren

- Kein spezifisches Tutorial, einfach genannte Dinge ausprobieren
- Vorschläge:
  - Kleine Bash scripte schreiben zum Beispiel zum erstellen von Orderstrukturen
  - Cheatsheet-Befehle anschauen, testen und nachvollziehen
  - Mit zwei ssh-Shells w und htop nachvollziehen
  - Programm mit tmux starten, ausloggen und wieder einloggen
  - Textumleitung, welcher Order ist der Größte?

- Wie auch vorhin: **Immer gerne Fragen** wenn etwas unklar ist! Allgemeines oder zu Infrastruktur!