**Cheat Sheet Systemnahe Programmierung**

**READ COMMAND:**

read -> read input from the user (cannot store any data, we need parameter)

Syntax: read <option> argument | Example: read input(hi) -> echo $input -> hi

read Heredoc **->** read var1 var2 << “Hello world”| $var1 ->“Hello”, $var2 = “world”

read -p (adds prompt text) -> read -p “Enter your username” username

read -s (hides user input) -> read -s password

read -n <number> (gives characters limit) -> read -n 4 input

read -a <arrayName> (save the input in array) -> read -a array <<< “Hello World” echo ${array[0]} => ”Hello” , echo ${array[1]} => “World”

**LS COMMAND:**

ls (list) -> Shows all files or directories in the current directory

ls /etc/ -> Switch to another directory and shows all files or directories

ls -l -> Long list format with more information. (It's given long list in which further information is given)

ls -m -> Print directories and files separated by a comma

ls -r -> Sort directories and files in the reverse order

ls -t -> Sort directories and files by modification time, with the most recently modified items

ls -X -> To sort directories and files alphabetically by the entry extension type

ls -a -> Shows all files including hidden ones. (you can’t see hidden files with “.”)

ls -R -> Lists contents of directories recursively, including subdirectories.

ls -lR -> Additional information on the directory tree, such as the file owner, size, and date and time of the last modification, type:

ls \*.txt -> Print the specific files type

ls -ltr -> Long listing format of files and directories with the latest modification date.

ls -lS -> Long listing format of files and directories sorted by file size, from largest to smallest.

ls -l /tmp -> Display Files Under Specific Directory.

ls -d -> Lists directories themselves, not their contents.

ls -1 -> Lists directories themselves, not their contents.

ls -f -> List all entries in directory order

**DATE COMMAND:**

Syntax: date [option] ... +[format] | Example: date -> Tue Oct 8 04:46:11 AM EDT 2024

date -d <string> -> display a specific date and time other than the current one

| date +%Y -> full Year | date +%m -> Month | date +%d -> day of the month |

| date +%H -> hour | date +%M -> minutes | date +%S -> seconds |

| date +%A -> full weekday name| date +%B -> Full month name | date --date=” yesterday” |

date -r -> Display Last Modified Timestamp of a Date File -> Example date -r /etc/hosts

alias -> Retrieve the date in preferred format without typing the full command every time

Example: alias [alias name]= ‘date [options]’

**COMMANDS FOR DIRECTORIES:**

mkdir <dirName> -> create a new directory

rmdir <dirName> -> remove directory (if is empty)

**COMMANDS FOR FILES:**

touch <filename> -> create a new file

cp <source> <destination> -> Copy FILE1 to FILE2

mv <currentFile> <newName> -> rename the file

rm -> delete a file

rm -rf -> remove the whole directory

cat (catalog) -> Shows the context of the text files. Always shows the whole file

Syntax: cat <filename>

cat <sourceFile> >> <destinationFile> -> Append file contents to another file.

head <filename> -> shows the first 10 lines in a file | tail <filename> -> last 10 lines in a file

wc -w <filename> -> Show the number of words, lines, and bytes in a file

wc -l <filename> -> shows the number of lines in a file

**SEARCHING COMMANDS:**

find <option> <path> <expression>

-name -> Searches for files or directories by name. (case-sensitive)

-not -> negates a condition

-empty -> Searches for empty files and directories

-maxdepth -> Limits the search to a specific depth

-print -> Displays the full path of each match. This is the default action

-perm -> Searches for files with specific permissions

-group -> Finds files owned by specific group.

-user -> Delivers files owned by specific user.

-ctime -> Searches for files with their status changed within a certain number of days.

-mtime -> Searches for files modified within a certain number of days.

-size -> finds files of a specific size. (c -> bytes, k -> kilobytes, M -> megabytes, G -> gigabytes)

-type -> Searches for files of a specific type (f -> files, d -> directories)

-iname -> Finds files or directories by names (case-insensitive)

-delete -> Delete files

**EXAMPLES:**

find ~ -name “\*.txt” -> ~ (current user’s directory), find all files that are .txt

find . -type f -name “\*.txt” -> . (searches the directory that you are currently in), type (file)

find . -type f -name “\*.txt” -delete -> Finds all files that are .txt and deletes them.

find $directory -maxdepth 1 -type f -name “\*.txt” -empty -> go in this directory, don’t go in the sub directories, search type file with the name “.txt” and check if it is empty or not.

find . -type f -exec grep -l “example” {} + -> -exec (allows to execute a command to each file), grep (searches for text pattern within the files) -l (instructs grep to list only the names of the files that contain the specific text, rather than displaying matching lines) {} (placeholder replaced by each file name that **find** matches) + (ends the execute command and indicates **find** should pass multiple filenames to grep at once)

find $directory -maxdepth 1 -type f -size +1M -> find files larger than one megabyte

find -name “example\*” -exec sed -i ‘s/[a-z]/5/g ‘ {} + -> replace the files example\* [a-z]/5

find "$dic" -maxdepth 1 -type f -user kristian -writable -> all files that are writable

find . -type f -mtime -30 -exec ls -lt {} + | sort -k6 -r -> sort by modification date

find $dic -maxdepth 1 -mtime 30+ -type f -name "\*" -exec rm {} \;

**CHECK IF FILE OR DIRECTORY EXISTS**

-f -> file

-d -> directory

-x (executable), -w (writable), -r (readable)

if [ ! -f $file ] -> checks if the file does not exist

**USER COMMANDS:**

who -> Show logged-in users

w -> Show logged-in users and their process

**PRINT COMMANDS:**

echo -> Print the text on the console

echo $? -> Print the exit code

**CONCANTANATION:**

| -> (the commands run parallel)

; -> they are executed in sequence

INPUT AND OUTPUT REDIRECTION:

< <filename> -> FILE Redirect input stream from the file -> Example: read hello (variable) < test.txt => result: hello = "Hello World"

> <filename> -> FILE Redirect output stream to the file -> Example: echo "Hello World1" > test.txt => test.txt contains "Hello World1”

>> <filename> -> FILE Redirect and append output stream -> Example: echo "Hello World2" >> test.txt => test.txt contains "Hello World1" and "Hello World2"

**ARGUMENTS:**

$# -> contains the number of arguments

$\* -> contains all arguments as one string.

$@ -> contains all arguments as string array

Example:

if [ $# -lt 2 ]; then

echo “Not enough arguments

fi

**VIM TEXTFILES**

**REGULAR EXPRESSIONS + egrep**

egrep + pattern + file

| [acf] -> matching specific characters | [a-e] -> matching specific range of characters. |

| \d -> digits [0-9] | \w -> digits, letter, underscore | \s -> whitespaces | \D no digits |

| ^ -> beginning of line | $ -> end of line | \n -> New Line | . -> all characters

**STRINGS:**

**REPLACE STRING:**

sed (stream editor) -i (tells sed to write the result to a file) ‘s (search) /<search regex>/<replacement regex>/g (global) ’ <filename>

Examples:

sed -i ‘s/bar/Linux/ ’ example.txt -> Replace first matching string in every line of the file

sed -i ‘s/bar/Linux/gI ’ example.txt -> replace all, i (ignore case)

sed -i ‘s/bar\b/linux/gI ‘ example.txt -> replace all but ignore the substring

sed -i ‘s/[A-Z]/5/g ‘ -> find all words that that have capital letters and replace it with 5

sed -i .bak ‘s/foo/Foo/g ‘ -> create backup file before overriding the existing one

**LENGTH OF STRING:**

${#string} -> it will return the length of the string

**STRING CONCATENATION**

+= -> concatenate string

Example: num1=”5” num2=”3” num1+=”5” -> result=$num1$num2 => result has „535“

**STRING COMPARISON**

[str1] = [str2] -> return true if both are equal

[ [str1] == [str2] ] -> return true if both are equal

-z [string] -> checks if the string is empty | -n [string] -> checks if the string is not empty

**Example:**

if [ “$str1” = “$str2 ]

if [[ “$str1” == “$str2 ]]

if [[ “$str1” == \*”$str2”\* ]] -> checks for substring, if str2 contains substring from str1

if [[ -z $str ]] -> if the string is empty

**REMOVE SUBSTRING:**

# -> Cut at the beginning

% -> Cut at the end

string : <position> : <length>

**Example:**

str=”Helloworld”

result=${str#Hello} -> return has world

echo ${str:2} -> lloworld | echo ${str:0:1} -> H | echo ${str:1:3} -> ell

echo ${str:1:$((${str}-2)} -> elloworl

**ARRAYS**

arr[3]=”hello” | “${arr[-1]}”

arr1=(“one” “two” “three”)

${#arr[\*]} -> length of an array

${arr[\*]} -> access all elements

arr2=($arr[\*]}) -> copy an array

${arr[@]:1} -> returns all except the first

${arr[@]:0:2} -> returns the first two elements

arr+=value -> add elements end of the array

${arr[@]::2} -> gives the first 2 elements from the array

IFS=’/’ read -ra array <<< “$path”

text\_files=$(find -type f -name "\*.txt")

**VARIABLES:**

readonly const=value -> constant

export -> make variable visible in other scripts

**ARITHMETIC:**

expr -> evaluates mathematic expression

Example: var=$(expr 2 + 3)

var=$((2+3))

float values -> bc

Example: num3=$(echo “$num1 \* $num2” |bc)

test -> expressions (true or false)

Strings: = != < >

Numbers: -eq -ne -lt -le -gt -ge

**LOOPS:**

for ((i=0; i<=9; i++)); do | for i in 0..9; do | for file int \*.txt; do

done done done

for name in $(find . -name \*.jpg); do

done

for file in "$1"/\*; do

if [ -f "$file" ] && [[ $file == \*.txt ]]

size=$(stat -c %s "$file")

if [ size > largest\_size ]

then

largest\_size=$size

largest\_file=$file

fi

fi

done

i=0

while [ $i -lt 10 ]; do

done

**SUBSTITUAION:**

${var:-value} | ${var-value} -> default time

if var is exist/set and not null -> use it

If var is exist/set but null -> use value

if var is not exist/set -> use value

${var:+value} | ${var+value} -> is variable set

if var exists and not null -> value

if var exists but null -> null

if var don’t exist -> null

${var:=value} | ${var=value} (redefine variables if not set)

if var exists and not null -> var

if var exists but null -> null

if var is not set -> assign and use value

${var:?value} (error messages)

if var is set and not null -> var

otherwise -> error code 1

**VIM MACRO:**

**record -> q + letter @a -> run macro**

**stop recording -> q @@ -> rerun last macro**