AWK

cheat.sheets:awk

# awk

# Pattern scanning and processing language

# Sum integers from a file or STDIN, one integer per line.

printf '1\n2\n3\n' | awk '{sum += $1} END {print sum}'

# Using specific character as separator to sum integers from a file or STDIN.

printf '1:2:3' | awk -F ":" '{print $1+$2+$3}'

# Print line number 12 of file.txt

awk 'NR==12' file.txt

# Print first field for each record in file.txt

awk '{print $1}' file.txt

Print only lines that match regex in file.txt

awk '/regex/' file.txt

# Print only lines that do not match regex in file.txt

awk '!/regex/' file.txt

# Print any line where field 2 is equal to "foo" in file.txt

awk '$2 == "foo"' file.txt

# Print lines where field 2 is NOT equal to "foo" in file.txt

awk '$2 != "foo"' file.txt

# Print line if field 1 matches regex in file.txt

awk '$1 ~ /regex/' file.txt

# Print line if field 1 does NOT match regex in file.txt

awk '$1 !~ /regex/' file.txt

# Print lines with more than 80 characters in file.txt

awk 'length > 80' file.txt

# Print a multiplication table.

awk -v RS='' '

{

for(i=1;i<=NF;i++){

printf("%dx%d=%d%s", i, NR, i\*NR, i==NR?"\n":"\t")

}

}

' <<< "$(seq 9 | sed 'H;g')"

# Specify output separator character.

printf '1 2 3' | awk 'BEGIN {OFS=":"}; {print $1,$2,$3}'

# Search paragraph for the given REGEX match.

# Paragraphs will be collapsed together.

awk -v RS='' '/42B/' file

# Search paragraph for the given REGEX match.

# Paragraphs will be separated with a new line.

awk -v RS= ORS='\n\n' '/42B/' file

# Display only first field in text taken from STDIN.

echo 'Field\_1 Field\_2 Field\_3' | awk '{print $1}'

# Note that in this case, you're far better off using cut(1).

# Use AWK solo; without the need for something via STDIN.

awk 'BEGIN {print("Example text.")}'

# Access environment variables from within AWK.

awk 'BEGIN {print ENVIRON["LS\_COLORS"]}'

# Count number of lines taken from STDIN.

free | awk '{L++} END {print(L)}'

# Cleaner, more efficient approach to the above.

free | awk 'END {print(NR)}'

# Output unique list of available sections under which to create a DEB package.

awk '!A[$1]++ {print($1)}' <<< "$(dpkg-query --show -f='${Section}\n')"

# Using process substitution (`<()` is NOT command substitution), with AWK and

# its associative array variables, we can print just column 2 for lines whose

# first column is equal to what's between the double-quotes.

awk '{NR != 1 && A[$1]=$2} END {print(A["Mem:"])}' <(free -h)

# While below is an easier and simpler solution to the above, it's not at all

# the same, and in other cases, the above is definitely preferable.

awk '/^Mem:/ {print($2)}' <(free -h)

# Output list of unique uppercase-only, sigil-omitted variables used in FILE.

awk '

{

for(F=0; F<NF; F++){

if($F~/^\$[A-Z\_]+$/){

A[$F]++

}

}

}

END {

for(I in A){

X=substr(I, 2, length(I))

printf("%s\n", X)

}

}

' FILE

# Output only lines from FILE between PATTERN\_1 and PATTERN\_2. Good for logs.

awk '/PATTERN\_1/,/PATTERN\_2/ {print}' FILE

# Pretty-print a table of an overview of the non-system users on the system.

awk -F ':' '

BEGIN {

printf("%-17s %-4s %-4s %-s\n", "NAME", "UID", "GID", "SHELL")

}

$3 >= 1000 && $1 != "nobody" {

printf("%-17s %-d %-d %-s\n", $1, $3, $4, $7)

}

' /etc/passwd

# Display the total amount of MiB of RAM available in the machine. This is also

# a painful but useful workaround to get the units comma-separated, as would be

# doable with Bash's own `printf` built-in.

awk '/^MemTotal:/ {printf("%'"'"'dMiB\n", $2 / 1024)}'

# It's possible to sort strings in AWK, as well as uniq-ing, meaning you can

# replace uniq(1) and sort(1) calls with just the one call of AWK. Granted, you

# can use `sort -u` to do both, but AWK offers much more functionality.

#

# Unlike when using AWK to uniq-ify, uniq(1) only works by adjacency, meaning

# the duplicate lines must be adjacent to one another for them to be handled.

awk '

{

!Lines[$0]++

}

END {

asorti(Lines, Sorted)

for (Line in Sorted) {

print(Sorted[Line])

}

}

' FILE

# Remove duplicate lines

awk '!seen[$0]++' file.txt

# Remove all empty lines

awk 'NF > 0' file.txt

cheat:awk

# To sum integers from a file or stdin, one integer per line:

printf '1\n2\n3\n' | awk '{ sum += $1} END {print sum}'

# To use a specific character as separator to sum integers from a file or stdin:

printf '1:2:3' | awk -F ":" '{print $1+$2+$3}'

# To print a multiplication table:

seq 9 | sed 'H;g' | awk -v RS='' '{for(i=1;i<=NF;i++)printf("%dx%d=%d%s", i, NR, i\*NR, i==NR?"\n":"\t")}'

# To specify an output separator character:

printf '1 2 3' | awk 'BEGIN {OFS=":"}; {print $1,$2,$3}'

# To execute commands only on rows that satisfy a certain condtion

printf "george jetson\nolive oyl\nbeetle bailey" | awk '$2=="bailey"{print $0}'

#

# To execute commands only on matching rows using regex

printf "george jetson\nolive oyl\nbeetle bailey" | awk '/ley$/{print $0}'

tldr:awk

# awk

# A versatile programming language for working on files.

# More information: <https://github.com/onetrueawk/awk>.

# Print the fifth column (a.k.a. field) in a space-separated file:

awk '{print $5}' path/to/file

# Print the second column of the lines containing "foo" in a space-separated file:

awk '/foo/ {print $2}' path/to/file

# Print the last column of each line in a file, using a comma (instead of space) as a field separator:

awk -F ',' '{print $NF}' path/to/file

# Sum the values in the first column of a file and print the total:

awk '{s+=$1} END {print s}' path/to/file

# Print every third line starting from the first line:

awk 'NR%3==1' path/to/file

# Print different values based on conditions:

awk '{if ($1 == "foo") print "Exact match foo"; else if ($1 ~ "bar") print "Partial match bar"; else print "Baz"}' path/to/file

# Print all lines where the 10th column value equals the specified value:

awk '($10 == value)'

# Print all the lines which the 10th column value is between a min and a max:

awk '($10 >= min\_value && $10 <= max\_value)'

COMM

cheat:comm

# Print lines appearing in both <file-1> and <file-2>:

comm -12 <(sort file1.csv) <(sort file2.csv)

# Print lines appearing only in <file-1>:

comm -23 <file-1> <file-2>

# For diffing CSVs from a database, consider:

# https://aswinkarthik.github.io/csvdiff/

tldr:comm

# comm

# Select or reject lines common to two files. Both files must be sorted.

# More information: <https://www.gnu.org/software/coreutils/comm>.

# Produce three tab-separated columns: lines only in first file, lines only in second file and common lines:

comm file1 file2

# Print only lines common to both files:

comm -12 file1 file2

# Print only lines common to both files, reading one file from `stdin`:

cat file1 | comm -12 - file2

# Get lines only found in first file, saving the result to a third file:

comm -23 file1 file2 > file1\_only

# Print lines only found in second file, when the files aren't sorted:

comm -13 <(sort file1) <(sort file2)

CUT

cheat:cut

# To cut out the third field of text or stdoutput that is delimited by a #:

cut -d# -f3

tldr:cut

# cut

# Cut out fields from `stdin` or files.

# More information: <https://www.gnu.org/software/coreutils/cut>.

# Print a specific character/field range of each line:

command | cut --characters|fields=1|1,10|1-10|1-|-10

# Print a range of each line with a specific delimiter:

command | cut --delimiter="," --fields=1

# Print a range of each line of the specific file:

cut --characters=1 path/to/file

DIFF

cheat:diff

# To view the differences between two files:

diff -u <file-1> <file-2>

# To view the differences between two directories:

diff -ur <dir-1> <dir-2>

# To ignore whitespace:

diff -ub <file-1> <file-2>

# To ignore blank lines:

diff -uB <file-1> <file-2>

# To ignore the differences between uppercase and lowercase:

diff -ui <file-1> <file-2>

# To report whether the files differ:

diff -q <file-1> <file-2>

# To report whether the files are identical:

diff -s <file-1> <file-2>

# To diff the output of two commands or scripts:

diff <(command1) <(command2)

# To generate a patch file from two files:

diff -Naur <file-1> <file-2> > <patch-file>

tldr:diff

# diff

# Compare files and directories.

# More information: <https://man7.org/linux/man-pages/man1/diff.1.html>.

# Compare files (lists changes to turn `old\_file` into `new\_file`):

diff old\_file new\_file

# Compare files, ignoring white spaces:

diff --ignore-all-space old\_file new\_file

# Compare files, showing the differences side by side:

diff --side-by-side old\_file new\_file

# Compare files, showing the differences in unified format (as used by `git diff`):

diff --unified old\_file new\_file

# Compare directories recursively (shows names for differing files/directories as well as changes made to files):

diff --recursive old\_directory new\_directory

# Compare directories, only showing the names of files that differ:

diff --recursive --brief old\_directory new\_directory

# Create a patch file for Git from the differences of two text files, treating nonexistent files as empty:

diff --text --unified --new-file old\_file new\_file > diff.patch

FOR

cheat:for

# basic loop

for i in 1 2 3 4 5 6 7 8 9 10

do

echo $i

done

# loop ls command results

for var in `ls -alF`

do

echo $var

done

# loop over all the JPG files in the current directory

for jpg\_file in \*.jpg

do

echo $jpg\_file

done

# loop specified number of times

for i in `seq 1 10`

do

echo $i

done

# same as above, but as one-liner

for i in `seq 1 10`; do echo $i; done

# loop specified number of times: the C/C++ style

for ((i=1;i<=10;++i))

do

echo $i

done

# loop specified number of times: the brace expansion

for i in {1..10}

do

echo $i

done

tldr:for

# for

# Perform a command several times.

# More information: <https://www.gnu.org/software/bash/manual/bash.html#Looping-Constructs>.

# Execute the given commands for each of the specified items:

for variable in item1 item2 ...; do echo "Loop is executed"; done

# Iterate over a given range of numbers:

for variable in {from..to..step}; do echo "Loop is executed"; done

# Iterate over a given list of files:

for variable in path/to/file1 path/to/file2 ...; do echo "Loop is executed"; done

# Iterate over a given list of directories:

for variable in path/to/directory1/ path/to/directory2/ ...; do echo "Loop is executed"; done

# Perform a given command in every directory:

for variable in \*/; do (cd "$variable" || continue; echo "Loop is executed") done

GREP

cheat:grep

# To search a file for a pattern:

grep <pattern> <file>

# To perform a case-insensitive search (with line numbers):

grep -in <pattern> <file>

# To recursively grep for string <pattern> in <dir>:

grep -R <pattern> <dir>

# Read search patterns from a file (one per line):

grep -f <pattern-file> <file>

# Find lines NOT containing pattern:

grep -v <pattern> <file>

# Set how many lines to show before (-B) and after (-A) pattern:

grep -B 3 -A 2 <pattern> <file>

# To grep with regular expressions:

grep "^00" <file> # Match lines starting with 00

grep -E "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}" <file> # Find IP add

# To find all files that match <pattern> in <dir>

grep -rnw <dir> -e <pattern>

# To exclude grep from your grepped output of ps:

# (Add [] to the first letter. Ex: sshd -> [s]shd)

ps aux | grep '[h]ttpd'

# Colour in red {bash} and keep all other lines

ps aux | grep -E --color 'bash|$'

tldr:grep

# grep

# Find patterns in files using regular expressions.

# More information: <https://www.gnu.org/software/grep/manual/grep.html>.

# Search for a pattern within a file:

grep "search\_pattern" path/to/file

# Search for an exact string (disables regular expressions):

grep --fixed-strings "exact\_string" path/to/file

# Search for a pattern in all files recursively in a directory, showing line numbers of matches, ignoring binary files:

grep --recursive --line-number --binary-files=without-match "search\_pattern" path/to/directory

# Use extended regular expressions (supports `?`, `+`, `{}`, `()` and `|`), in case-insensitive mode:

grep --extended-regexp --ignore-case "search\_pattern" path/to/file

# Print 3 lines of context around, before, or after each match:

grep --context|before-context|after-context=3 "search\_pattern" path/to/file

# Print file name and line number for each match with color output:

grep --with-filename --line-number --color=always "search\_pattern" path/to/file

# Search for lines matching a pattern, printing only the matched text:

grep --only-matching "search\_pattern" path/to/file

# Search `stdin` for lines that do not match a pattern:

cat path/to/file | grep --invert-match "search\_pattern"

HEAD

cheat:head

# To show the first 10 lines of <file>:

head <file>

# To show the first <number> lines of <file>:

head -n <number> <file>

# To show the first <number> bytes of <file>:

head -c <number> <file>

tldr:head

# head

# Output the first part of files.

# More information: <https://www.gnu.org/software/coreutils/head>.

# Output the first few lines of a file:

head -n count path/to/file

IF

# if

# Performs conditional processing in shell scripts.

# See also: `test`, `[`.

# More information: <https://www.gnu.org/software/bash/manual/bash.html#Conditional-Constructs>.

# Execute the specified commands if the condition command's exit status is zero:

if condition\_command; then echo "Condition is true"; fi

# Execute the specified commands if the condition command's exit status is not zero:

if ! condition\_command; then echo "Condition is true"; fi

# Execute the first specified commands if the condition command's exit status is zero otherwise execute the second specified commands:

if condition\_command; then echo "Condition is true"; else echo "Condition is false"; fi

# Check whether a [f]ile exists:

if [[ -f path/to/file ]]; then echo "Condition is true"; fi

# Check whether a [d]irectory exists:

if [[ -d path/to/directory ]]; then echo "Condition is true"; fi

# Check whether a file or directory [e]xists:

if [[ -e path/to/file\_or\_directory ]]; then echo "Condition is true"; fi

# Check whether a variable is defined:

if [[ -n "$variable" ]]; then echo "Condition is true"; fi

# List all possible conditions (`test` is an alias to `[`; both are commonly used with `if`):

man [

JOIN

Top of Form

$ curl cheat.sh/

Bottom of Form

# join

# Join lines of two sorted files on a common field.

# More information: <https://www.gnu.org/software/coreutils/join>.

# Join two files on the first (default) field:

join file1 file2

# Join two files using a comma (instead of a space) as the field separator:

join -t ',' file1 file2

# Join field3 of file1 with field1 of file2:

join -1 3 -2 1 file1 file2

# Produce a line for each unpairable line for file1:

join -a 1 file1 file2

# Join a file from `stdin`:

cat path/to/file1 | join - path/to/file2

PASTE

cheat:paste

# To concat columns from files:

paste <file>...

# To list the files in the current directory in three columns:

ls | paste - - -

# To combine pairs of lines from a file into single lines:

paste -s -d '\t\n' <file>

# To number the lines in a file, similar to nl(1):

sed = <file> | paste -s -d '\t\n' - -

# To create a colon-separated list of directories named bin, suitable for use in the PATH environment variable:

find / -name bin -type d | paste -s -d : -

tldr:paste

# paste

# Merge lines of files.

# More information: <https://www.gnu.org/software/coreutils/paste>.

# Join all the lines into a single line, using TAB as delimiter:

paste -s path/to/file

# Join all the lines into a single line, using the specified delimiter:

paste -s -d delimiter path/to/file

# Merge two files side by side, each in its column, using TAB as delimiter:

paste file1 file2

# Merge two files side by side, each in its column, using the specified delimiter:

paste -d delimiter file1 file2

# Merge two files, with lines added alternatively:

paste -d '\n' file1 file2

$

SED

cheat.sheets:sed

# sed

# A stream editor. Used to perform basic text transformations

# Preview a file edit, via substitution.

sudo sed 's/Name=Xfce Session/Name=Xfce\_Session/' FILE

# Replace the same string more than once per line (g flag)

sudo sed 's/Name=Xfce Session/Name=Xfce\_Session/g' FILE

# Edit a file (adding -i flag), in-place; changes are made to the file(s).

sudo sed -i 's/Name=Xfce Session/Name=Xfce\_Session/' FILE

# It can become necessary to escape special characters in your string.

sed -i 's/\/path\/to\/somewhere\//\/path\/to\/anotherplace\//' FILE

# Change your sed delimiter to a pipe to avoid escaping slashes.

sed -i 's|/path/to/somewhere/|/path/to/anotherplace/|' FILE

cheat:sed

# To replace all occurrences of "day" with "night" and write to stdout:

sed 's/day/night/g' <file>

# To replace all occurrences of "day" with "night" within <file>:

sed -i 's/day/night/g' <file>

# To replace all occurrences of "day" with "night" on stdin:

echo 'It is daytime' | sed 's/day/night/g'

# To remove leading spaces:

sed -i -r 's/^\s+//g' <file>

# To remove empty lines and print results to stdout:

sed '/^$/d' <file>

# To replace newlines in multiple lines:

sed ':a;N;$!ba;s/\n//g' <file>

# To insert a line before a matching pattern:

sed '/Once upon a time/i\Chapter 1'

# To add a line after a matching pattern:

sed '/happily ever after/a\The end.'

tldr:sed

# sed

# Edit text in a scriptable manner.

# See also: `awk`, `ed`.

# More information: <https://www.gnu.org/software/sed/manual/sed.html>.

# Replace all `apple` (basic regex) occurrences with `mango` (basic regex) in all input lines and print the result to `stdout`:

command | sed 's/apple/mango/g'

# Execute a specific script [f]ile and print the result to `stdout`:

command | sed -f path/to/script.sed

# Print just a first line to `stdout`:

command | sed -n '1p'

SORT

cheat.sheets:sort

# sort

# Sort lines of text files

# Return the contents of the British English dictionary, in reverse order.

sort -r /usr/share/dict/british-english

# The GNU sort(1) command can also filter out adjacent duplicate lines and can

# therefore overlap with the uniq(1) command. However, uniq(1) has some options

# that sort(1) cannot do so refer to the man page for you situation if you

# require something beyond a basic uniqueness check. In addition, there is the

# potential for parallizing the processing by piping sort(1) into uniq(1) for

# non trivial tasks.

#

# By default, sort(1) sorts lines or fields using the ASCII table. Here, we're

# essentially getting alphanumeric sorting, where case is handled separately; -

# this results in these words being adjacent to one another, thus duplicates

# are removed.

#

# If you need better uniq-ing, you could refer to AWK & its associative arrays.

printf '%s\n' this is a list of of random words with duplicate words | sort -u

# Sort numerically. If you don't provide the `-n` flag, sort(1) will instead

# sort by the ASCII table, as mentioned above, meaning it'll display as 1, 10, -

# 11, 2, 3, 4, etc.

printf '%d\n' {1..9} 10 11 | sort -n

# You can even sort human-readable sizes. In this example, the 2nd column is

# being sorted, thanks to the use of the `-k` flag, and the sorting is

# reversed, so that the top-most storage space hungry filesystems are displayed

# from df(1).

df -ht ext4 /dev/sd[a-z][1-9]\* | sed '1d' | sort -rhk 2

cheat:sort

# To sort a file:

sort <file>

# To sort a file by keeping only unique:

sort -u <file>

# To sort a file and reverse the result:

sort -r <file>

# To sort a file randomly:

sort -R <file>

# To sort a file and store the output in another file:

sort <inputFile> -o <outputFile>

# Sort by default uses /var/tmp to store temp files but size of /var/tmp directory is limited. In order to sort huge use a directory with adequate size:

sort -T <tempDirectory> <file>

tldr:sort

# sort

# Sort lines of text files.

# More information: <https://www.gnu.org/software/coreutils/sort>.

# Sort a file in ascending order:

sort path/to/file

# Sort a file in descending order:

sort --reverse path/to/file

# Sort a file in case-insensitive way:

sort --ignore-case path/to/file

# Sort a file using numeric rather than alphabetic order:

sort --numeric-sort path/to/file

# Sort `/etc/passwd` by the 3rd field of each line numerically, using ":" as a field separator:

sort --field-separator=: --key=3n /etc/passwd

# Sort a file preserving only unique lines:

sort --unique path/to/file

# Sort a file, printing the output to the specified output file (can be used to sort a file in-place):

sort --output=path/to/file path/to/file

# Sort numbers with exponents:

sort --general-numeric-sort path/to/file

SSH

cheat.sheets:ssh

# ssh

# OpenSSH SSH client (remote login program)

# SSH in via PEM file, which normally needs 0600 permissions.

ssh -i /path/to/file.pem user@example.com

# Connect through a non-standard port. It's recommended not to use the default

# port of 22, as it is so often targeted, due to it being so commonplace.

ssh -p 2222 user@example.com

# Connect and forward the authentication agent.

ssh -A user@example.com

# Execute a command on a remote server.

ssh -t user@example.com 'the-remote-command'

# Tunnel an X session over SSH, via X11 Forwarding.

ssh -X user@example.com

# Redirect traffic with a tunnel between local host (port 8080) and a remote

# host (remote.example.com:5000) through a proxy (personal.server.com).

ssh -f -L 8080:remote.example.com:5000 user@personal.server.com -N

# Launch a specific X application over SSH.

ssh -X -t user@example.com 'chromium-browser'

# Create a SOCKS proxy on localhost and port 9999.

ssh -D 9999 user@example.com

# Connect to server, but allow for X11 forwarding, while also using Gzip

# compression (can be much faster; YMMV), and using the `blowfish` encryption.

# For more information, see: http://unix.stackexchange.com/q/12755/44856

ssh -XCc blowfish user@example.com

# Copy files and directories, via SSH, from remote host to the current working

# directory, with Gzip compression. An option for when `rsync` isn't available.

#

# This works by creating (not temporary!) a remote Tar archive, then piping its

# output to a local Tar process, which then extracts it to STDOUT.

ssh user@example.com 'tar -C /var/www/Shared/ zcf - asset1 asset2' | tar zxf -

# Explicitly specify a key for connection. Useful if you have too many

# authentication failures for a given username.

ssh -i some\_id\_rsa -o IdentitiesOnly=yes them@there:/path/

# Temporarily disable `pubkey` authentication for this instance.

ssh -o PubkeyAuthentication=no username@hostname.com

# Mount a remote directory or filesystem, through SSH, to a local mount point.

# Install SSHFS from: https://github.com/libfuse/sshfs

sshfs name@server:/path/to/folder /path/to/mount/point

# EMACS can read files through SSH. Below, is a link to related documentation.

#

# http://www.gnu.org/software/emacs/manual/html\_node/emacs/Remote-Files.html

#

emacs /ssh:name@server:/path/to/file

# Get help for SSH escape sequences. Useful for terminating unresponsive

# sessions. The default escape character is ~ (tilde), escapes are only

# recognized immediately after a newline.

$ <Enter>~?

cheat:ssh

---

tags: [ ssh ]

---

# To ssh via pem file (which normally needs 0600 permissions):

ssh -i <pemfile> <user>@<host>

# To connect on a non-standard port:

ssh -p <port> <user>@<host>

# To connect and forward the authentication agent:

ssh -A <user>@<host>

# To execute a command on a remote server:

ssh -t <user>@<host> 'the-remote-command'

# To connect to a host with a specific key exchange algorithm:

# Full list of available algorithms : man ssh\_config

ssh -oKeXAlgorithms=+diffie-hellman-group-exchange-sha1 <user>@<server>

# To tunnel an x session over SSH:

ssh -X <user>@<host>

# Redirect traffic with a tunnel between local host (port 8080) and a remote

# host (remote.example.com:5000) through a proxy (personal.server.com):

ssh -f -L 8080:remote.example.com:5000 user@personal.server.com -N

# To launch a specific x application over SSH:

ssh -X -t <user>@<host> 'chromium-browser'

# To create a SOCKS proxy on localhost and <port>:

ssh -qND <port> <user>@<host>

# To tunnel an ssh session over the SOCKS proxy on localhost and port 9999:

ssh -o "ProxyCommand nc -x 127.0.0.1:9999 -X 4 %h %p" <user>@<host>

# -X use an xsession, -C compress data, "-c blowfish" use the encryption blowfish:

ssh <user>@<host> -C -c blowfish -X

# For more information, see:

# http://unix.stackexchange.com/q/12755/44856

# To copy files and folders through ssh from remote host to pwd with tar.gz

# compression when there is no rsync command available:

ssh <user>@<host> "cd /var/www/Shared/; tar zcf - asset1 asset2" | tar zxf -

# To mount folder/filesystem through SSH

# Install SSHFS from https://github.com/libfuse/sshfs

# Will allow you to mount a folder securely over a network.

sshfs <user>@<host>:/path/to/folder /path/to/mount/point

# Emacs can read file through SSH

# Doc: http://www.gnu.org/software/emacs/manual/html\_node/emacs/Remote-Files.html

emacs /ssh:<user>@<host>:<file>

tldr:ssh

# ssh

# Secure Shell is a protocol used to securely log onto remote systems.

# It can be used for logging or executing commands on a remote server.

# More information: <https://man.openbsd.org/ssh>.

# Connect to a remote server:

ssh username@remote\_host

# Connect to a remote server with a specific identity (private key):

ssh -i path/to/key\_file username@remote\_host

# Connect to a remote server using a specific port:

ssh username@remote\_host -p 2222

# Run a command on a remote server with a [t]ty allocation allowing interaction with the remote command:

ssh username@remote\_host -t command command\_arguments

# SSH tunneling: Dynamic port forwarding (SOCKS proxy on `localhost:1080`):

ssh -D 1080 username@remote\_host

# SSH tunneling: Forward a specific port (`localhost:9999` to `example.org:80`) along with disabling pseudo-[T]ty allocation and executio[N] of remote commands:

ssh -L 9999:example.org:80 -N -T username@remote\_host

# SSH jumping: Connect through a jumphost to a remote server (Multiple jump hops may be specified separated by comma characters):

ssh -J username@jump\_host username@remote\_host

# Agent forwarding: Forward the authentication information to the remote machine (see `man ssh\_config` for available options):

ssh -A username@remote\_host

TAIL

cheat:tail

# To show the last 10 lines of <file>:

tail <file>

# To show the last <number> lines of <file>:

tail -n <number> <file>

# To show the last lines of <file> starting with <number>:

tail -n +<number> <file>

# To show the last <number> bytes of <file>:

tail -c <number> <file>

# To show the last 10 lines of <file> and to wait for <file> to grow:

tail -f <file>

tldr:tail

# tail

# Display the last part of a file.

# See also: `head`.

# More information: <https://www.gnu.org/software/coreutils/tail>.

# Show last 'count' lines in file:

tail --lines count path/to/file

# Print a file from a specific line number:

tail --lines +count path/to/file

# Print a specific count of bytes from the end of a given file:

tail --bytes count path/to/file

# Print the last lines of a given file and keep reading file until `Ctrl + C`:

tail --follow path/to/file

# Keep reading file until `Ctrl + C`, even if the file is inaccessible:

tail --retry --follow path/to/file

# Show last 'num' lines in 'file' and refresh every 'n' seconds:

tail --lines count --sleep-interval seconds --follow path/to/file

TEST

# test

# Check file types and compare values.

# Returns 0 if the condition evaluates to true, 1 if it evaluates to false.

# More information: <https://www.gnu.org/software/coreutils/test>.

# Test if a given variable is equal to a given string:

test "$MY\_VAR" == "/bin/zsh"

# Test if a given variable is empty:

test -z "$GIT\_BRANCH"

# Test if a file exists:

test -f "path/to/file\_or\_directory"

# Test if a directory does not exist:

test ! -d "path/to/directory"

# If A is true, then do B, or C in the case of an error (notice that C may run even if A fails):

test condition && echo "true" || echo "false"

WC

cheat:wc

# To count the number of words (file or STDIN):

wc -w <file>

cat <file> | wc -w

# To count the number of lines (file or STDIN):

wc -l <file>

cat <file> | wc -l

# To count the number of bytes (file or STDIN):

wc -c <file>

cat <file> | wc -c

# To count files and directories at a given location:

ls | wc -l

# To if you ever use `wc` in a shell script and need to compare the output with an int you can

# clean the output (wc returns extra characters around the integer) by using xargs:

ls -l | wc -l | xargs

tldr:wc

# wc

# Count lines, words, and bytes.

# More information: <https://www.gnu.org/software/coreutils/wc>.

# Count all lines in a file:

wc --lines path/to/file

# Count all words in a file:

wc --words path/to/file

# Count all bytes in a file:

wc --bytes path/to/file

# Count all characters in a file (taking multi-byte characters into account):

wc --chars path/to/file

# Count all lines, words and bytes from `stdin`:

find . | wc

# Count the length of the longest line in number of characters:

wc --max-line-length path/to/file

WHILE

# while

# Simple shell loop.

# More information: <https://manned.org/while>.

# Read `stdin` and perform an action on every line:

while read line; do echo "$line"; done

# Execute a command forever once every second:

while :; do command; sleep 1; done