

/ /

```
ls -dl * | sed 's/-[^-]*$//'  
find . -maxdepth 1 -type d -not -path '.' | sed 's@./@@;s@-.*@@'
```

LC_COLLATE

```
### Toggle LANG C.UTF-8 <-> ko_KR.UTF-8  
function tg-lang() {  
# LANG  
case "$LANG" in  
ko_KR*)  
export LANG=C.UTF-8  
;;  
*)  
export LANG=ko_KR.UTF-8  
;;  
esac  
# LC_ALL  
unset LC_ALL  
  
# PS1  
if [ -f "/etc/profile.d/_ps1" ]; then  
source /etc/profile.d/_ps1  
fi  
  
echo -e "Locale toggled to: \033[1;32m$LANG\033[0m"  
}  
## alias  
# alias tl='tg-lang' -> /etc/profile.d/10-alias.sh
```

```
#!/usr/bin/env bash  
  
## Get Package Name  
if [[ -r $PWD/PKG-INFO ]]; then  
PKG_NAME=$(grep -i '^Name:' PKG-INFO | sed 's/[nN]ame:[[:space:]]*  
//' | tr -d '\r')  
else  
echo "PKG-INFO not found. Need package name for install"  
read -p "Enter Package Name: " PKG_NAME  
fi  
  
## Permission Check
```

```

PERM_ELEV=""
[[ $UID -ne 0 ]] && PERM_ELEV="sudo"

## Clean Previous Build
rm -rf dist

## Build and Install
pip3 wheel -w dist --no-cache-dir --no-build-isolation --no-deps $PWD

${PERM_ELEV} pip3 install --no-index --find-links dist "${PKG_NAME}"

```

./configure --docdir=

```

#!/usr/bin/env bash

##                                0
DEPTH=${1:-0}

##      가
if ! [[ "$DEPTH" =~ ^[0-9]+$ ]]; then
    echo "Usage: $0 [number]"
    exit 1
fi

##
ORIGINAL_PWD=$(pwd)
TARGET_PATH=$ORIGINAL_PWD

#      ( )
for (( i=0; i<$DEPTH; i++ )); do
    # Stop when reach the top directory
    if [ "$TARGET_PATH" = "/" ]; then
        break
    fi
    TARGET_PATH=$(dirname "$TARGET_PATH")
done

##
DOCDIR_VALUE=$(basename "$TARGET_PATH")

##      ( )      가 "/"
##      "/usr/share/doc/"
##
if [ "$DOCDIR_VALUE" = "/" ] || [ -z "$DOCDIR_VALUE" ]; then
    TARGET_PATH=$ORIGINAL_PWD
    RECOVER_DEPTH=$(( DEPTH > 0 ? 1 : 0 ))

    for (( i=0; i<$RECOVER_DEPTH; i++ )); do
        TARGET_PATH=$(dirname "$TARGET_PATH")
    done
fi

```

```

done
DOCDIR_VALUE=$(basename "$TARGET_PATH")
fi

##
echo "/usr/share/doc/${DOCDIR_VALUE}"

unset DEPTH ORIGINAL_PWD TARGET_PATH RECOVER_DEPTH DOCDIR_VALUE

```

./configure --docdir=

```

#!/usr/bin/env bash

##
##
## : ./configure --prefux=usr --docdir=$(docdir 1)
if [ -z $1 ]; then
    DOCDIR_VALUE=$(basename $(pwd))
else
    DOCDIR_VALUE=$(basename $(dirname $(pwd)))
fi

echo "/usr/share/doc/${DOCDIR_VALUE}"
unset DOCDIR_VALUE

```

LFS mount and chroot script v.0.2

lfs-dive.sh

```

#!/usr/bin/env bash
# if [[ -z ${LFS} ]]; then echo "Enter the LFS 13.0" | is same follow
: ${LFS:=/mnt/LFS}
echo "Base path - ${LFS}"
export LFS

if [[ $UID == 0 ]]; then USER_ELEV=""; else USER_ELEV="sudo"; fi

# Source code packages Repo. mount
mountpoint -q $LFS/sources || ${USER_ELEV} mount n40l:/SOURCES/LFS-13.0
/mnt/LFS/sources

# 1. If already mounted virtual kernel file system then execute chroot

```

```

script.
# 2. if need mount virtual kernel file system.

if mountpoint -q $LFS/dev; then
    lfs-redive.sh
    exit 0
else
    ${USER_ELEV} mount -v --bind /dev $LFS/dev
    ${USER_ELEV} mount -vt devpts devpts -o gid=5,mode=0620 $LFS/dev/pts
    ${USER_ELEV} mount -vt proc proc $LFS/proc
    ${USER_ELEV} mount -vt sysfs sysfs $LFS/sys
    ${USER_ELEV} mount -vt tmpfs tmpfs $LFS/run
    if [ -h $LFS/dev/shm ]; then
        ${USER_ELEV} install -v -d -m 1777 $LFS$(realpath /dev/shm)
    else
        ${USER_ELEV} mount -vt tmpfs -o nosuid,nodev tmpfs $LFS/dev/shm
    fi
    lfs-redive.sh
fi

```

lfs-redive.sh

```

if [[ ! -f "${LFS}"/stage.3 ]]; then echo "Not stage 3, no need chroot";
exit 0; fi
if [[ $UID == 0 ]]; then USER_ELEV=""; else USER_ELEV="sudo"; fi

${USER_ELEV} chroot "$LFS" /usr/bin/env -i \
    HOME=/root \
    TERM="$TERM" \
    PS1='(lfs chroot) \u:\w\$ ' \
    PATH=/usr/local/bin:/usr/bin:/usr/sbin \
    MAKEFLAGS="-j$(nproc)" \
    TESTSUITEFLAGS="-j$(nproc)" \
    /bin/bash --login

```

Host LFS chroot v.01

가 . 6

:
auto-lfs s : chroot ,
auto-lfs c : chroot



LFS, LFS_PART /boot, /boot/efi, /home, /sources

#!/bin/bash

```
# LFS 12.1-systemd
# <7.2 >
# 가 /
# Begin ~/bin/lfs-env-config

VERSION=0.1.7
LFS=/mnt/lfs
LFS_PART=/dev/nvme0n1p8
LFS_BOOT=/dev/nvme0n1p6
LFS_EFI=/dev/nvme0n1p2
LFS_HOME=/dev/nvme0n1p7

#
if ! mountpoint -q $LFS;          then mount $LFS_PART $LFS; fi
if ! mountpoint -q $LFS/boot;    then mount $LFS_BOOT $LFS/boot; fi
if ! mountpoint -q $LFS/boot/efi; then mount $LFS_EFI $LFS/boot/efi; fi
if ! mountpoint -q $LFS/home;    then mount $LFS_HOME $LFS/home; fi
if ! mountpoint -q $LFS/sources; then mount /mnt/nfs/sources $LFS/sources;
fi

# /dev
if [ ! -d $LFS/dev ]; then
    echo "Make necessary directories"
    mkdir -pv $LFS/{dev,proc,sys,run}
else
    echo "All required directories have been verified."
fi

#          chroot          exit          가
case $1 in
    clean|c|-c)
        if mountpoint -q $LFS/dev/shm; then
            # Unmount Kernel Virtual File System used in previous LFS
sessions.
            echo "LFS is now set to '$LFS'"
            echo "Now proceed with the unmount of <Kernel Virtual File
System> for chroot"
            for kvfs in dev/shm dev/pts sys proc run dev ; do
                if mountpoint $LFS/$kvfs; then umount -q $LFS/$kvfs; fi
            done
            findmnt -D -R $LFS
            echo "Ok unmount unnecessary devices"
            echo "Good bye"
            exit 0
        else
            echo "Cleanup of Kernel Virtual File System is Complete."
            findmnt -D -R $LFS
            exit 0
        fi
    ;;
    start|s|-s)

```

```

# Mount the Kernel Virtual File System to Enter LFS chroot
environment.
echo "Now proceed with the mount of <Kernel Virtual File System> for
chroot"
echo "LFS is now set to '$LFS'"
if ! mountpoint -q $LFS/dev;      then mount --bind /dev $LFS/dev;
fi
if ! mountpoint -q $LFS/dev/pts; then mount -t devpts devpts -o
gid=5,mode=0620 $LFS/dev/pts; fi
if ! mountpoint -q $LFS/proc;    then mount -t proc proc $LFS/proc;
fi
if ! mountpoint -q $LFS/sys;     then mount -t sysfs sysfs $LFS/sys;
fi
if ! mountpoint -q $LFS/run;     then mount -t tmpfs tmpfs $LFS/run;
fi
if ! mountpoint -q $LFS/dev/shm; then mount -t tmpfs -o nosuid,nodev
tmpfs $LFS/dev/shm;      fi
if ! mountpoint -q $LFS/sources; then mount --rbind /mnt/nfs/sources
$LFS/sources;           fi
findmnt -D -R $LFS
# Run 'chroot'
chroot "$LFS" $(type -p env) -i \
    HOME=/root           \
    TERM="$TERM"         \
    PS1='<LFS> \u@\h [ \w ] \$ ' \
    PATH=/usr/bin:/usr/sbin \
    MAKEFLAGS="-j$(nproc)" \
    TESTSUITEFLAGS="-j$(nproc)" \
    NINJAJOBS="$(nproc)" \
    /bin/bash --login
exit 0
;;
version|v|-v)
    echo "LFS chroot script $VERSION"
    exit 0
;;
help|h|-h)
    echo "LFS chroot script $VERSION"
    echo "Usage:"
    echo "clean    or c- Unmount Kernel Virtual File System"
    echo "start     or s- Mount and run 'chroot' command with necessary
variables"
    echo "version  or v- Print script version"
    echo "help     or h - This screen"
    exit 0
;;
*)
    #Guidance on the factors needed to execute script
    echo "sh ./lfs-env-config [ start | clean | help | version ]"
    exit 0
;;

```

```
esac
## End ~/bin/lfs-env-config
```

가

cd 가 .

```
alias <filename>='source <filename>'
```

:

가

1. URL Git 가 가
2. 가.zip 가 가
3. 가 가
4. 가 가
- 5.

가

1. alias
2. wget, git, gawk

1. git clone source .

```
user@host:[ ~ ] $ pre-stage
https://xorg.freedesktop.org/archive/individual/proto/xcb-proto-1.17.0.tar.x
z
xcb-proto-1.17.0.tar 100%
[=====>] 148.19K 302.76KB/s
[Files: 1 Bytes: 148.19K [153.40KB/s] Redirects:
0 Todo: 0 Errors: 0 ]
user@host:[ /workbench/xcb-proto-1.17.0 ] $ _
```

```
#!/bin/bash
```

```
Green='\033[0;32m'
BIGreen='\033[1;32m'
Red='\033[0;31m'
```

```
Yellow='\033[0;33m'
Color_Off='\033[0m'

if [ "$(type -t p4c)" != 'alias' ] ; then
    echo -e "${Yellow}p4c is not registered${Color_Off}, so the
functionality is not available."
    source ${HOME}/.bash_aliases
    return 2>&/dev/null
fi

if [ -z "$1" ]; then echo -e "${Green}An appropriate factor (file name or
download link) is required.${Color_Off}"; return; fi
InputStringType=$(echo "$1" | awk 'BEGIN { FS = ":" } ; { print $1 }')
BuildBase="$LFS"/workbench
SourceBase="$LFS"/sources
TargetFile=$(basename "$1")
GitDirCheck=$(basename -s .git "$1")
GitCheck=${TargetFile: -3}
ZipCheck=${TargetFile: -3}
TargetDir=$(echo "$TargetFile" | sed -e 's/\.tar.*$//' -e 's/\.tgz*$//' -e
's/\.src*$//' -e 's/\.zip*$//' -e 's@^.*/@@')
NotFoundMsg1="The directory derived from the filename is
${BIGreen}$TargetDir${Color_Off}.\n\
However, it does not exist, so it will move to the ${Yellow}most recent
directory created${Color_Off}.\n\
Please check if it is correct."

if [ $(stat -c %u $BuildBase) -ne $(id -u) ] ; then echo -e "Check
${Red}$BuildBase${Color_Off} Ownership"; return; fi

function chg_dir() {
    if [ -d "$1" ]; then
        cd "$1"
    else
        cd $(ls -tcA -w1 | head -n1)
        echo -e "$NotFoundMsg1"
    fi
}

# Change directory "$BuildBase"
if [ "$PWD" != "$BuildBase" ]; then
    chg_dir "$BuildBase"
fi

## Handling Git
if [ "$GitCheck" == "git" ] ; then
    if [ -d "$GitDirCheck" ] ; then
        chg_dir "$GitDirCheck"
        return
    else
        pushd $SourceBase
```

```

        if [ -d $GitDirCheck ] ; then
            rm -rf $GitDirCheck
        fi
        git clone "$1"
        cp -R $GitDirCheck $BuildBase
        popd
#     echo "Git #1"
    chg_dir "$GitDirCheck"
    return
fi
fi

case "$InputStringType" in
    https|http|ftp)
        wget --no-verbose -nc --directory-prefix="$SourceBase" "$1"
        ;;
    *)
        ;;
esac

# Prepare Package Build
if [ "$ZipCheck" == "zip" ] ; then
    mkdir -p "$TargetDir"
    chg_dir "$TargetDir"
    unzip -q "$SourceBase"/"$TargetFile"
    return
else
    if [ ! -d "$TargetDir" ]; then
        tar -xf "$SourceBase"/"$TargetFile" 2>/dev/null
    fi
    chg_dir "$TargetDir"
fi

unset InputStringType BuildBase SourceBase TargetFile GitDirCheck GitCheck
ZipCheck TargetDir NotFoundMsg1

```

AI :

```

#     (#! /bin/bash) source     가

# 1. tput
BOLD=$(tput bold)
BLUE=$(tput setaf 4)
GREEN=$(tput setaf 2)
BIGREEN="${BOLD}${GREEN}"
RED=$(tput setaf 1)
YELLOW=$(tput setaf 3)
RESET=$(tput sgr0)

```

```

# OpenRC
print_status() {
    local msg="$1"
    local status=$2
    local col=$(tput cols)
    printf " ${BLUE}${BOLD}*${RESET} %s" "$msg"
    tput hpa $((col - 7))
    [ $status -eq 0 ] && printf "[  ${GREEN}OK${RESET}  ]\n" || printf "[
${RED}!!${RESET}  ]\n"
}

# 2.          (p4c          )
if [ "$(type -t p4c)" != 'alias' ]; then
    echo -e "p4c is not ${YELLOW}alias registered${RESET}, so functionality
is unavailable."
    # source
    [ -f "${HOME}/.bash_aliases" ] && . "${HOME}/.bash_aliases"
    return 2 2>/dev/null
fi

# 3.
if [ -z "$1" ]; then
    echo -e "${GREEN}An appropriate factor (file name or download link) is
required.${RESET}"
    return
fi

#          (LFS          )
BuildBase="${LFS}/workbench"
SourceBase="${LFS}/sources"
TargetFile=$(basename "$1")
TargetDir=$(echo "$TargetFile" | sed -r
's/\.\tar(\.)*?$/\.\tgz$/\.\src$/\.\zip$//;s/\.\git$//')
InputScheme=$(echo "$1" | grep -oP '^[^:]+(?:=:?)') # http, https, ftp

# 4.
if [ ! -w "$BuildBase" ]; then
    echo -e "Check ${RED}$BuildBase${RESET} Ownership/Write Permission"
    return
fi

# 5.          (가          )
chg_dir() {
    if [ -d "$1" ]; then
        cd "$1"
    else
        local latest=$(ls -tcA -w1 | head -n1)
        cd "$latest"
        echo -e "Derived: ${BIGREEN}$1${RESET} (Not found)"
        echo -e "Moved to latest: ${YELLOW}$latest${RESET}"
    fi
}

```

```
#
[ "$PWD" != "$BuildBase" ] && cd "$BuildBase"

# 6.                (http, https, ftp)
case "$InputScheme" in
  http|https|ftp)
    wget --no-verbose -nc -P "$SourceBase" "$1"
    print_status "Downloading $TargetFile" $?
    ;;
esac

# 7.                (Git vs Archive)
if [[ "$1" == *.git ]] || [ "$InputScheme" == "git" ]; then
  # Git
  GitName=$(basename -s .git "$1")
  if [ -d "$GitName" ]; then
    chg_dir "$GitName"
  else
    pushd "$SourceBase" >/dev/null
    [ -d "$GitName" ] && rm -rf "$GitName"
    git clone "$1" && cp -R "$GitName" "$BuildBase/"
    popd >/dev/null
    chg_dir "$GitName"
    print_status "Git Clone: $GitName" $?
  fi
fi
else
  #
  if [[ "$TargetFile" == *.zip ]]; then
    mkdir -p "$TargetDir" && cd "$TargetDir"
    unzip -q "$SourceBase/$TargetFile"
    print_status "Unzip: $TargetFile" $?
  else
    if [ ! -d "$TargetDir" ]; then
      tar -xf "$SourceBase/$TargetFile" 2>/dev/null
      print_status "Extract: $TargetFile" $?
    fi
    chg_dir "$TargetDir"
  fi
fi

#                (local                source                unset)
unset BOLD BLUE GREEN BIGREEN RED YELLOW RESET BuildBase SourceBase
TargetFile TargetDir InputScheme GitName
```

From:
<http://wiki.gamu.kr/dokuwiki/> -

Permanent link:
<http://wiki.gamu.kr/dokuwiki/linuxfromscratch/auto-lfs>

Last update: **2026/04/15 17:58**



