```
#!/bin/bash

# HELPER FUNCTIONS #

# Create an absolute symlink from two or more relative paths
# Allows you to create multiple symlinks at once as with mv or cp.
# E.g symlink target_dir *.fq
symlink () {
        local to="$1" # Target directory
        shift 1

        # Get absolute target path
        local to_fp=$( readlink -f "$to" )

        # creates symlinks for each argument
        for f in "$@"
        do
                local from_fp=$(readlink -f "$f") # full path of file to be linked

                #create symlink
                ln -f -s "$from_fp" "$to_fp"

                if [[ "$?" -ne 0 ]]
                then
                        return "$?"
                fi
        done
        return 0
}

## If file is compressed: decompress to output directory
## Otherwise just symlink it.
decompress () {
        local out_dir="$1" # target directory
        shift

        for f in "$@"
        do
                local suffix="${f##*.}" #filetype of input file

                if [[ "$suffix" == "gz" ]]
                then
                        local prefix="${f%%.gz}" #filename with .gz
                        gunzip -c "$f" > "${out_dir}/${prefix}" #unzip file

                elif [[ "$suffix" == "fq" || "$suffix" == "fastq" ]]
                then

                        # if file isn't compressed then just create a link
                        symlink "$out_dir" "$f"

                else
                        printf "Error: %s is not a valid sequence file \n" "$1"
                        return 1
                fi
        done
        return 0
}

# Returns the prefix of a read file of the form '<prefix>_<number>.fq'
prefix () {
        local f_name="$1"
        echo "${f_name%%?.fq*}"
        return 0
}

# Returns all reads within the current directory as a string.
```

```
get_reads () {

        # Find all .fq/.fq.gz files in current directory
        if test -n "$(find "." -maxdepth 1 -name '*.fq*' -print -quit)"
        then
                eval echo "*.fq*" #return list of files
                return 0
        else
                echo '' # return empty string if none found
                return 1
        fi
}

# Given a file name, finds the paired file in the same directory. (Files must have same prefix)
# Only works if files are in the format '<prefix>_<integer>.<suffix>
get_pair () {
        local f_name="$1"       #name of file to be paired
        local f_prefix=$(prefix "$f_name")      # prefix of file
        local f_suffix="${f_name#*.}"   # suffix
        local in_dir=$(dirname "$f_name")       # Directory to search for paired file
        local paired=( $(basename ${in_dir}/${f_prefix}2.${f_suffix}) ) #Get paired filename

        #Check if pair was found
        if [[ -z "$paired" ]]
        then
        # If no paired file found: return an empty string
                echo ''
                return 1
        else
        #if found, return the pair
                echo "$paired"
                return 0
        fi
}

# Given two directories (dir1, dir2) move all files from dir1 to dir2. In cases of name-conflict, concaten
ate the files
# This is useful for merging to sequence files from the same individual with different sets of reads
merge_to () {

        dir="$1" # output directory
        shift

        for f in "$@"
        do
                if [[ -f "${dir}/${f}" ]]
                then
                        # If file exists in output directory: concatenate
                        cat "$f" >> "${dir}/${f}"
                else
                        # If file does not already exist: copy
                        cp "$f" "$dir"
                fi
        done
        return 0
}

## Check that user inputted either 2 sequence files or a directory of files
check_args () {

        #Check for presence of adapters
        if [[ ! -f "$1" || "${1##*.}" != "fa" ]]
        then
                return 3
        fi

        #Check number of inputs is 1 or 2
```

```
        if [[ "$#" -gt 3  || "$#" -lt 2 ]]
        then
                return 1
        fi

        #Check whether argument is a directory or a pair of files
        if [[ -d "$2" && "$#" -eq 2 ]]
        then
                # Argument is a directory of seq files
                echo "d"
                return 0

        elif [[ -f "$2" && -f "$3" ]]
        then
                # Argument is a pair of seq files
                echo "f"
                return 0
        else
                return 2
        fi

}

# QUALITY CONTROL MODULES #

#Generate fastqc reports for every input file
# Usage: fastqc_reports <output_directory> <read1> ... <readn>
fastqc_reports () {
        local report_dir="$1"   # Output directory for fastqc reports
        shift 1

        # Loop through arguments and generate fastqc reports
        for i in "$@"
        do
                fastqc -o "$report_dir" -t 48 "$i" >> "/dev/null" 2>&1

                if [[ "$?" -ne 0 ]]
                then
                        return 1
                fi
        done
        return 0
}

## Check pairing information is correct. If jumbled, repair pairing information.
# Usage: check_pairing <logfile_directory> <paired_output_directory> <unpaired_output_directory> <read1
> ... <readn>
check_pairing () {
        local logdir="$1" # logfile directory
        local pd="$2"   # Directory for paired output
        local upd="$3"  # Directory for unpaired output

        # If the first pairing check has already been completed. Create a new logfile
        if [ -e "${logdir}/pairing_check.txt" ]
        then
                local log="${logdir}/pairing_check2.txt"
        else
                local log="${logdir}/pairing_check.txt"
        fi

        shift 3

        #Find one of each pair of files
        #NOTE: This only works if paired files have _1 and _2 suffixes - Edit this block if suffix is some
thing else
        for f in "$@"
        do
```

```
                # If "f" is the forward sequence (1) then find its paired file
                if [[ ! "${f/"_1."}" == "$f" ]]
                then
                        local pair=$(get_pair "$f")

                        #Repair pairing information and add result to the logdir
                        pairfq_lite makepairs -f "$f" -r "$pair" -fp "${pd}/$f" -rp "${pd}/$pair" -fs "${u
pd}/$f" -rs "${upd}/$pair" --stats \
                        >> "$log" 2>&1

                        # If all files are correctly paired, pairfq_lite will create empty unpaired files.
                        # These should be removed.
                        if [ -s "${upd}/${f}" ]
                        then
                                rm "${upd}/${f}"
                        fi

                        if [ -s "${upd}/${pair}" ]
                        then
                                rm "${upd}/${pair}"
                        fi
                fi
        done

        return 0
}

# Trim adapter sequences using the inputted adapter file.
# Also trim low quality trailing bases ('Illumina low quality segments')
# Can be run on single reads using pair_mode="u" and on paired reads using "p"
# Paired end usage: trim p <adapter_file> <logfile_directory> <paired_output_directory> <unpaired_output_d
irectory> <read1> ... <readn>
# Unpaired usage: trim u <adapter_file> <logfile_directory> <output_directory> <read1> ... <readn>
trim () {
        local pair_mode=$1 # p or u
        local ad_file="$2" # file containing adapter sequences
        local logdir="$3" # logfile directory

        # These settings can be tinkered with to optimize performance for the data set (see trimmomatic ma
nual)
        local allowed_mismatch=2
        local palindromeClipThreshold=30
        local simpleClipThreshold=10
        local min_adapter_length=5

        # Paired Mode
        if [[ "$pair_mode" == "p" ]]
        then

                local pd="$4"   # Output directory for paired files
                local upd="$5" # Output directory for unpaired files
                shift 5

                # Loop through input files, trimming adapter
                for f in "$@"
                do
                        if [[ ! "${f/"_1."}" == "$f" ]]
                        then
                                local pair=$(get_pair "$f")

                                trimmomatic PE -threads 48 -phred64 -trimlog "${logdir}/paired_trimlog.txt
" "$f" "$pair" "$pd/$f" "$upd/$f" "$pd/$pair" "$upd/$pair" \
                                 "ILLUMINACLIP:$ad_file:$allowed_mismatch:$palindromeClipThreshold:$simple
ClipThreshold:$min_adapter_length" \
                                >> "/dev/null" 2>&1

                                # If empty files are created, remove them.
```

```
                                if [ -s "${upd}/${f}" ]
                                then
                                        rm "${upd}/${f}"
                                fi

                                if [ -s "${upd}/${pair}" ]
                                then
                                        rm "${upd}/${pair}"
                                fi

                                if [[ "$?" -ne 0 ]]
                                then
                                        return 1
                                fi
                        fi
                done

        # Unpaired Mode
        elif [[ "$pair_mode" == "u" ]]
        then

                local upd="$4"  # Output directory
                shift 4

                # Loop through input files, trimming adapter
                for f in "$@"
                do
                        trimmomatic SE -threads 48 -trimlog "${logdir}/unpaired_trimlog.txt" "$f" "$upd/$f
" \
                        "ILLUMINACLIP:$ad_file:$allowed_mismatch:$palindromeClipThreshold:$simpleClipThres
hold:$min_adapter_length" \
                        >> "/dev/null" 2>&1

                        if [ -s "${upd}/${f}" ]
                        then
                                rm "${upd}/${f}"
                        fi

                        if [[ "$?" -ne 0 ]]
                        then
                                return 1
                        fi
                done
        fi
        return 0
}


##---------  MAIN -----------##

# All global vars preceded with 'g'
gmode=$(check_args "$@") # Check arguments. Input mode (directory/files) will be returned

gad_file=$(readlink -f "$1")    # File containing known adapter sequences;
gpdir=$(dirname "$2")   # Directory containing paired files
cd "$gpdir"

# If incorrect arguments inputted, output error message
case "$?" in
1 )
        printf "Error: Incorrect number of arguments\n"
        printf "Usage: 'daph_pipeline <PATH_TO/ADAPTER_SEQUENCES> </PATH_TO/SEQUENCE_DIRECTORY> OR </PATH_
TO/SEQ_FILE1> </PATH_TO/SEQ_FILE2>'\n"
        exit 1
        ;;
2 )
        printf "Error: Incorrect arguments\n"
```

```
        printf "Usage: 'daph_pipeline <PATH_TO/ADAPTER_SEQUENCES> </PATH_TO/SEQUENCE_DIRECTORY> OR </PATH_
TO/SEQ_FILE1> </PATH_TO/SEQ_FILE2>'\n"
        exit 1
        ;;
3 )
        printf "Error: %s is not a valid adapter file: Check trimmomatic manual \n" "$gad_file"
        exit 1
        ;;
esac

# Transform input into an array of sequence file names
if [[ "$gmode" == "d" ]]
then
        printf "Will begin pipeline on %s\n" "$2"
        gpaired_files=( $(get_reads) )  # Seq files within input directory
elif [[ "$gmode" == "f" ]]
then
        printf "Will begin pipeline on %s and %s\n" "$2" "$3"
        gpaired_files=("${@:2}")        # Read files
fi

#create directories for final output files
mkdir -p "final/paired/fastqc"
mkdir -p "final/unpaired/fastqc"
mkdir -p "final/flash/fastqc"
mkdir -p "final/logs"

# remove all previous logfiles
rm -f "final/logs/*"

gfin_paired=$(readlink -f "final/paired")
gfin_unpaired=$(readlink -f "final/unpaired")
glogdir=$(readlink -f "final/logs") # Directory where logs will be stored

# Create initial fastqc reports
printf "Generating fastqc reports... \n"

mkdir -p "fastqc"
if ! fastqc_reports "fastqc" "${gpaired_files[@]}"
then
        printf "Error: Fastqc reports could not be completed. Check input files.\n"
        exit 1
fi

#create directory for process tree
mkdir -p "processed"

#If files are compressed, decompress them to new directory. Otherwise just create symlinks
printf "Preparing sequence files for processing... \n"
decompress "processed" "${gpaired_files[@]}"

cd "processed"
gpdir="$PWD"
ginitdir=$(readlink -f "$gpdir")
gpaired_files=( $(get_reads) )

# Check that pairing information is intact
printf "Checking pairing information...\n"

mkdir -p "paired" && mkdir -p "unpaired"

gpdir=$( readlink -f "paired" )
gudir=$( readlink -f "unpaired" )

check_pairing "$glogdir" "$gpdir" "$gudir" "${gpaired_files[@]}"

cd "$gpdir"
```

```
gpaired_files=( $(get_reads) )

# If no paired files are present at this point, there is something wrong: exit with error
if [ ${#gpaired_files[@]} -eq 0 ]
then
        printf "Error: No paired files found. Pipeline cannot be continued. \n"
        exit 1
fi

mkdir -p "rm_adapt/paired" && mkdir -p "rm_adapt/unpaired"

printf "Trimming adapters... \n"

# Trim adapters from paired reads
if ! trim "p" "$gad_file" "$glogdir" "rm_adapt/paired" "rm_adapt/unpaired" "${gpaired_files[@]}"
then
        printf "Error: files could not be trimmed. Check input. \n"
        exit 1
fi

cd "rm_adapt"
split="$PWD"

cd "$gudir"
gunpaired_files=( $( get_reads ) )

if [[ ! ${#gunpaired_files[@]} -eq 0 ]]
then
        mkdir -p "tmp"          # unpaired files will be trimmed and then merged with other unpaired file
s; This directory will be deleted after merging
        if ! trim "u" "$gad_file" "$glogdir" "tmp" "${gunpaired_files[@]}"
        then
                printf "Error: files could not be trimmed. Check input. \n"
                exit 1
        fi

        # Copy files from the temporary unpaired folder to the other unpaired folder.
        # In cases where two files have the same name, concatenate the files.

        printf "Merging unpaired reads...\n"

        cd "tmp"
        tmp_files=( $(get_reads) )
        merge_to "${split}/unpaired" "${tmp_files[@]}"

        # Remove temp folder
        cd ..
        rm -rf "tmp"
fi

cd "$split"
gpdir=$( readlink -f "paired" )
gudir=$( readlink -f "unpaired" )
cd "$gpdir"
gpaired_files=( $(get_reads) )

printf "Pairing check... \n"

mkdir -p "paired" && mkdir -p "unpaired"

split="$PWD"
check_pairing "$glogdir" "paired" "unpaired" "${gpaired_files[@]}"
cd "paired"
gpdir="$PWD"
gpaired_files=( $( get_reads ) )

cd "$gudir"
```

```
tmp_files=( $( get_reads ) )
merge_to "${split}/unpaired" "${tmp_files[@]}"

cd "${split}/unpaired"
gudir="$PWD"
gunpaired_files=( $( get_reads ) )

# Copy finished files to final directories

cd "$gudir"
cp "${gunpaired_files[@]}" "$gfin_unpaired"

cd "$gpdir"
cp "${gpaired_files[@]}" "$gfin_paired"

printf "Generating final reports...\n"

cd "$gudir"
if ! fastqc_reports "${gfin_unpaired}/fastqc" "${gunpaired_files[@]}"
then
        printf "error: final fastqc reports could not be generated \n"
        exit 1
fi

cd "$gpdir"

if ! fastqc_reports "${gfin_paired}/fastqc" "${gpaired_files[@]}"
then
        printf "error: final fastqc reports could not be generated \n"
        exit 1
fi

cd "$gcombined"
if ! fastqc_reports "${gfin_flash}/fastqc" "${gflashed[@]}"
then
        printf "error: final fastqc reports could not be generated \n"
        exit 1
fi

# By default processed tree is removed after use. Just comment out next line if you want to keep intermedi
ate files.
rm -rf "$ginitdir"
exit 0
```