Master solutions

Your first script

```
#!/bin/bash
echo 'Hello, World!'
```

Count to 100

Make sure the parameter expansion is quoted properly. It may not be necessary here, but it is good practice.

```
#!/bin/bash
for i in {1..100}
do
    echo "$i"
done
```

Here, we use brace expansion, so we can't use quotes.

```
#!/bin/bash
echo {1..100}
```

Set up a script directory for your user

The following line should be added to either ~/.profile or ~/.bashrc. You have to log out and back in to make it work.

```
export PATH="$PATH:/home/user/scripts"
```

You can test whether it worked by running

```
echo "$PATH"
```

in your terminal.

Parse some options

```
#!/bin/bash
while getopts 'h?abc:' opt; do
    case "$opt" in
    h|\?)
        echo 'Available options: -h, -a, -b, -c ARGUMENT'
        exit 0
        ;;
    a)
        echo 'Option a selected'
        ;;
    b)
        echo 'Option b selected'
        ;;
    c)
        echo "Option f selected with argument $OPTARG"
        ;;
    esac
done

shift $((OPTIND-1))
```

The shift line at the end is important! If you have more arguments (that aren't options), you can't access them otherwise.

Output all your arguments

```
#!/bin/bash
for i in "$@"
do
    echo "$i"
done
```

Make sure all quotes are put correctly. Especially the expression "$@" MUST be quoted.

Find big files

```bash
#!/bin/bash

usage() {
    echo "usage: $0 directory"
    exit 1
}

# We want exactly one argument
if [[ "$#" -ne 1 ]]; then
    usage
fi

dir="$1"

# Test whether argument is a directory
if ! [[ -d "$dir" ]]
then
    usage
fi

# Find files in dir, sort by size and print largest 5
find "$dir" -printf '%s %p\n' | sort -nr | head -n 5
```

The use of a function for `usage` is not required, but it's good practice.

Self-reproducing script

```bash
#!/bin/bash
cp "$0" backup.sh
```

This second example also works, but is bad practice: it is discouraged to use `cat` to get the contents of a file in a bash script. There are almost always better ways to do that.

```bash
#!/bin/bash
cat "$0" > backup.sh
```

Make your bash prompt fancy

Not much to say here, all solutions will be individual

Maze generator

`RANDOM` is an environment variable that takes on a different numeric value every time you read it.

```bash
#!/bin/bash
while true
do
    (( $RANDOM % 2 ))  \
        && echo -n '/' \
        || echo -n '\' \
    sleep 0.07
done
```

Simple backup script

The `find` option `-type f` makes sure only files are listed (and not directories). The `-ctime -1` option makes sure only files not older than 1 day (24 hours) are listed.

```bash
#!/bin/bash
find . -type f -ctime -1 | xargs tar -czf 'backup.tar.gz'
```

Screen brightness control

```bash
#!/bin/bash

(( $# == 0 )) && {
    echo "No argument given; exiting"
    exit 1;
}

# -z tests whether string is empty (zero length). We search for + or - in the string.
if [[ -z $( echo $1 | grep -Eo "\+|-" ) ]]
then
    newbrightness="$1"
```

```
else
    brightness="$(cat /sys/class/backlight/intel_backlight/brightness)"
    newbrightness="$(( $brightness $1 ))" # becomes for example 10 +1
fi

# prevent screen from going all black. The threshold value might vary,
# since the scale of brightness values is different on every system
if (( $newbrightness < 10 ))
then
    newbrightness=10
fi

# use tee instead of redirections because we need sudo
echo "$newbrightness" | sudo tee /sys/class/backlight/intel_backlight/brightness > /dev/null
```

The file path might of course vary. If you don't find a brightness file in /sys/class/backlight, chances are this script is not possible on your system.

Automatic update script

This script is designed to be used with sudo, i.e. sudo updatescript. If you use it like that, you don't need sudo for zypper inside the script.

```
#!/bin/bash
# First, move old files out of the way safely. Discard error messages.
mv -b /home/user/updatelog /home/user/updatelog-old > /dev/null 2>&1

{ zypper up > /home/user/updatelog } \
    && rm /home/user/updatelog

poweroff
```

Of course, the zypper command is for openSUSE, and other distros require other commands.

Disable/Enable external monitor output

These are some examples. It depends on your setup what you need.

```
#!/bin/bash
xrandr --output HDMI-0 --auto --left-of LVDS-0
```

```
#!/bin/bash
xrandr --output HDMI-0 --auto
xrandr --output LVDS-0 --off
```

```
#!/bin/bash
# detect a monitor
if xrandr | grep 'HDMI-0 connected' > /dev/null
then
    xrandr --output HDMI-0 --right-of LVDS-0
fi
```

Display a random headline from Reddit

This exercise requires you to parse strings to display them nicely. That can be quite tedious, and the commands used for it are often cryptic (cf. sed). Don't be scared.

```
#!/bin/bash
curl -A 'Mozilla/5.0 (Windows; U; MSIE 7.0; Windows NT 6.0; en-US)' -s
    https://www.reddit.com/r/showerthoughts.json \
    | jshon -e data -e children -a -e data -e title \
    | shuf -n 1 \
    | sed 's/\\//g' \
    | sed 's/^"//' \
    | sed 's/"$//'
```

The three seds are merely for pretty printing. The first removes backslashes, the second removes the first quote and the last removes the last quote.

In a sed expression, ^ matches the beginning of a line and $ matches the end. Also, backslashes (among other characters) have to be *escaped* using yet another backslash.

Wallpaper change every 10 minutes

You can run this script in the background, like this: `./wallpaperscript &`

```bash
#!/bin/bash
while true
do
    feh --bg-fill --randomize ~/pictures/wallpaper/*
    sleep $(( 60 * 10 ))
done
```

Your own TODO program

```bash
#! /bin/bash
# use a subshell to contain the directory change
(
    # we work in .config, this isn't mandatory. ~/.config is where many
    # programs store user configuration.
    mkdir -p ~/.config/todo
    cd ~/.config/todo
    touch todolist # touch creates a file if it doesn't exist yet

    while getopts 'i:d:h' opt; do
        case "$opt" in
            h|\?)
                echo 'Valid flags: -i -d -h'
                ;;
            i)
                echo "$OPTARG" >> todolist
                ;;
            d)
                # sed '3d' deletes the 3rd line from a file.
                sed -e "${OPTARG}d" todolist > /tmp/todolist
                cp /tmp/todolist todolist
                ;;
        esac
    done
    # the -n option adds line numbers
    cat -n todolist
)
```

Reminder script

```bash
#!/bin/bash
mins="$1"
shift 1

echo notify-send "'$@'" | at now + "$mins" min &>/dev/null
echo "Reminder: $@ set for $mins mins from now."
```

Youtube downloader

```bash
#!/bin/bash

query="$@"

# get youtube search results page
# search for links that contain 'watch\?v=' followed by 11 characters
# take the first one
ytid=$( \
    curl -s --data-urlencode "search_query=$query" https://www.youtube.com/results \
    | grep -oE 'watch\?v=.{11}' \
    | head -n 1 \
)

echo "$ytid"

youtube-dl -f bestaudio -o "%(title)s_%(id)s.%(ext)s" "https://www.youtube.com/$ytid"
```

Web radio

```bash
#!/bin/bash

# tidy up first
killall mpv

# The & is not required but useful. Cache size is increased for lag free streaming
mpv http://213.251.190.165:9000 -cache 1024 &
```

Image resizer

```bash
#!/bin/bash

mkdir -p resized

for i in *
do
    # The > indicates that ImageMagick should only shrink larger images
    convert "$i" -resize '100x100>' resized/"$i"
done
```

Assignment-fetcher

```bash
#!/bin/bash

while getopts 'h?lf:u:p:' opt; do
    case "$opt" in
    h|\?)
        echo "usage: $0 [-l] [-f FILTER] [-u USER] [-p PASSWORD] URL"
        exit 0
        ;;
    f)
        filter="$OPTARG"
        ;;
    u)
        user="$OPTARG"
        ;;
    p)
        pass="$OPTARG"
        ;;
    l)
        list=true
    esac
done

shift $((OPTIND-1))

url=$1


# Bare url up to the first slash.
# The grep first finds all characters up to the first dot, and then from there on finds
# all characters that are NOT slashes up to the first slash.
# The sed is used to escape all slashes within the url. / becomes \/
baseurl=$( \
    echo $url \
    | grep -oe '.*\.[^/]*/' \
    | sed -e 's/\//\\\//g' \
)

# URL without file name (up to the last slash).
# The grep just finds everything up to a slash. By default it uses greedy matching,
# which means that it finds the longest fitting sequence.
# The sed is used to escape all slashes within the url.
longurl=$( \
    echo $url \
    | grep -oe '.*/' \
    | sed -e 's/\//\\\//g' \
)

# The last command used in the command chain below depends on an option.
# if -l was passed, we just echo the found pdfs. Otherwise download them.
if [ $list == true ]
then
    # the -n option is for xargs
    lastcmd="-n 1 echo"
else
    lastcmd="wget --user $user --password $pass"
fi

# First, find strings that end in .pdf
# Then, prepend the base url to all strings that are relative urls
# Then, apply the filter
# Then, if an url starts with // (relative to root), prepend the longurl
# last, download or display the urls
curl -s $url \
    | grep -oe '[^"]*\.pdf' \
    | sed -e "s/^\//$baseurl\//" \
    | grep -e "/[^/]*$filter[^/]*\.pdf" \
    | awk "{ if ( \$0 !~ /\/\// ) { print \"$longurl\" \$0 } else {print} }" 2>/dev/null \
    | xargs $lastcmd
```