Woolsey Workshop

Cheatsheet: Raspberry Pi Basics

==

General

==

General Information

- Raspberry Pi products are officially developed by *Raspberry Pi (Trading) Ltd*, a wholly owned subsidiary of the *Raspberry Pi Foundation* headquartered in Cambridge, UK.
- The Raspberry Pi is a fully functional computer based on the ARM architecture that can run various operating systems.
- *Raspberry Pi OS* (previously called *Raspbian*) is the officially recommended OS for Raspberry Pi systems. It is free, based on the *Debian* Linux distribution, and is optimized for Raspberry Pi hardware.
- The *Raspberry Pi Imager* application is used to install a fresh OS image on a Raspberry Pi. It can be downloaded from the Raspberry Pi OS page with installation instructions on the Getting Started page. The imager application provides the recommended *Raspberry Pi OS* and includes choices for installing many other operating systems. It even has the ability to set up your WiFi network credentials (so that WiFi is enabled upon your first log in) along with other configuration options.
- The OS and all files are stored on a microSD card. For best performance, the card should be rated as class A1 with a minimum size of 8 GB.
- Many programming languages come preinstalled on *Raspberry Pi OS* and many more can be installed.
- The *Raspberry Pi OS* default username and password are typically *pi* and *raspberry* respectively.

Resources

- Raspberry Pi Foundation (charitable organization)
- Raspberry Pi (Trading) Ltd (hardware and software)
- Raspberry Pi Documentation (official guides and general documentation)
- Raspberry Pi OS (download installer)
- Setting Up Your Raspberry Pi (installation and basic configuration)
- Using Your Raspberry Pi (configuration and operation)
- Remote Access (from another computer)
- Woolsey Workshop Raspberry Pi Tutorials

Common Desktop GUI Operations

- Shut down the system.
  Raspberry Menu > Shutdown…
- Open the *Terminal* application.
  Raspberry Menu > Accessories > Terminal

- Install and manage software packages.
  Raspberry Menu > Preferences > Recommended Software (install and manage recommended software)
  Raspberry Menu > Preferences > Add / Remove Software (install and manage other software)
- Change system wide configurations.
  Raspberry Menu > Preferences > Raspberry Pi Configuration
  - Change your user password in the **System** tab.
  - Enable on-board interfaces, e.g. SSH, I2C, SPI, etc., in the **Interfaces** tab.

Common Command Line Operations

- Access privileged commands (when permission is denied).
  Syntax: `$ sudo <command>`
- Get the local network IP address of the Raspberry Pi.
  Computer: `$ ping raspberrypi.local`
  Raspberry Pi: `$ hostname -I`
- Configure settings.
  `$ sudo raspi-config`
- Copy a file from your computer to the Raspberry Pi (run on your computer).
  Syntax: `$ scp <file> <rp_user>@raspberrypi.local:<rp_path>`
  Example: `$ scp blink.py pi@raspberrypi.local:~/src/python`
- Copy a directory from your computer to the Raspberry Pi (run on your computer).
  Syntax: `$ scp -r <dir> <rp_user>@raspberrypi.local:<rp_path>`
  Example: `$ scp -r python pi@raspberrypi.local:~/src`
- Copy a file from the Raspberry Pi to your computer (run on your computer).
  Syntax: `$ scp <rp_user>@raspberrypi.local:<rp_path> <file>`
  Example: `$ scp pi@raspberrypi.local:~/src/python/blink.py .`
- Copy a directory from the Raspberry Pi to your computer (run on your computer).
  Syntax: `$ scp -r <rp_user>@raspberrypi.local:<rp_path> <dir>`
  Example: `$ scp -r pi@raspberrypi.local:~/src/python .`
- Display the current directory structure.
  `$ tree`
- Download a file from the internet.
  Syntax: `$ wget <url>`
  Example: `$ wget https://datasheets.raspberrypi.com/rpi4/raspberry-pi-4-datasheet.pdf`
- Change your password.
  `$ passwd`
- Get the user and group IDs for the current user.
  `$ id`

- Show the groups to which you belong.
  ```
  $ groups
  ```
- Display the current CPU load, RAM usage, and running system processes.
  ```
  $ top
  ```
- Show the Raspberry Pi model.
  ```
  $ cat /proc/device-tree/model
  ```
- Show OS version information.
  ```
  $ uname -a
  ```
- Show memory usage.
  ```
  $ free -h
  ```
- Display system messages.
  ```
  $ dmesg
  ```

Software Package Management With The Advanced Package Tool (APT)

- List installed packages.
  ```
  $ apt list --installed
  ```
- List upgradable packages.
  ```
  $ sudo apt update && apt list --upgradable
  ```
- Upgrade the installed packages.
  ```
  $ sudo apt update && sudo apt upgrade
  ```
- Upgrade the entire OS.
  ```
  $ sudo apt update && sudo apt full-upgrade
  $ sudo reboot
  ```
- Search for a package.
  Syntax: `$ apt search <package>`
- Show a package's information.
  Syntax: `$ apt show <package>`
- Install a package.
  Syntax: `$ sudo apt install <package>`
- Remove a package.
  Syntax: `$ sudo apt remove <package>`

==

Hardware & GPIO Pins

==

General Information

- Most Raspberry Pi boards have a 40-pin header that contains two 5 volt and two 3.3 volt supply pins along with 26 usable digital general purpose input/output (GPIO) pins. All GPIO pins are rated at 3.3 V and are not tolerant to higher voltages.
- Expansion boards that follow the official Raspberry Pi specification are called HATs (Hardware Attached on Top). Others are called pHAT, Bonnet, Cap, etc.
- Verify which pin numbering scheme is being used within instructions or tutorials so that you connect your circuits properly and don't accidentally damage your Raspberry Pi. The following three schemes are typically used: *Broadcom* (BCM), *Board* or *Physical* (header location), and *WiringPi* (deprecated).
- The Raspberry Pi Pin-Out website is a great resource for understanding GPIO functionality and numbering schemes.
- In order to avoid accidental damage to the Raspberry Pi pins or connected hardware,
  - connect all hardware before connecting power and booting up the Raspberry Pi,
  - reset GPIO pins back to their default (input) state whether you are working with the pins from the command line or from within programs, and
  - shutdown and disconnect power from the Raspberry Pi before disconnecting or changing hardware.

Power Pins

- The two 5 V supply pins are rated at approximately 300 mA (maximum supply current across both pins). The available current is limited by the power usage of all active interfaces across the entire board.
- The two 3.3 V supply pins are rated at 50 mA (maximum supply current across both supply pins).

Digital I/O Pins

- Rated 3.3 V @ 16 mA (maximum of 50 mA across all GPIO pins simultaneously).
- On power-on reset, all GPIO pins revert to general-purpose inputs with default pull-ups (*GPIO0-8*) or pull-downs (*GPIO9-GPIO27*) applied.
- Any GPIO pin can be configured as an input or output.
- Any GPIO pin can be configured as an interrupt.
- Pins *GPIO2* and *GPIO3* have fixed pull-up resistors (used for I2C communication). All other GPIO pins can be configured as pull-up or pull-down with approximately 50 KΩ resistance.
- Hardware PWM is available on *GPIO12*, *GPIO13*, *GPIO18*, and *GPIO19*. Software PWM is available on all GPIO pins.

- If more current is required on a GPIO output, a transistor or relay can be used to drive the component powered by an external power supply.

Pin-Out

1. **3V3** - 3.3 V Power Supply
2. **5V** - 5 V Power Supply
3. **GPIO2/I2C1_SDA** - Digital I/O 2 / I2C1 SDA (fixed pull-up)
4. **5V** - 5 V Power Supply
5. **GPIO3/I2C1_SCL** - Digital I/O 3 / I2C1 SCL (fixed pull-up)
6. **GND** - Ground
7. **GPIO4** - Digital I/O 4
8. **GPIO14/UART0_TXD** - Digital I/O 14 / UART0 TXD
9. **GND** - Ground
10. **GPIO15/UART0_RXD** - Digital I/O 15 / UART0 RXD
11. **GPIO17/SPI1_CE1** - Digital I/O 17 / SPI1 CE1
12. **GPIO18/SPI1_CE0/PWM0** - Digital I/O 18 / SPI1 CE0 / Pulse Width Modulation 0
13. **GPIO27** - Digital I/O 27
14. **GND** - Ground
15. **GPIO22** - Digital I/O 22
16. **GPIO23** - Digital I/O 23
17. **3V3** - 3.3 V Power Supply
18. **GPIO24** - Digital I/O 24
19. **GPIO10/SPI0_MOSI** - Digital I/O 10 / SPI0 MOSI
20. **GND** - Ground
21. **GPIO9/SPI0_MISO** - Digital I/O 9 / SPI0 MISO
22. **GPIO25** - Digital I/O 25
23. **GPIO11/SPI0_SCK** - Digital I/O 11 / SPI0 SCK
24. **GPIO8/SPI0_CE0** - Digital I/O 8 / SPI0 CE0
25. **GND** - Ground
26. **GPIO7/SPI0_CE1** - Digital I/O 7 / SPI0 CE1
27. **GPIO0** - Reserved (EEPROM Communication)
28. **GPIO1** - Reserved (EEPROM Communication)
29. **GPIO5** - Digital I/O 5
30. **GND** - Ground
31. **GPIO6** - Digital I/O 6
32. **GPIO12/PWM0** - Digital I/O 12 / Pulse Width Modulation 0
33. **GPIO13/PWM1** - Digital I/O 13 / Pulse Width Modulation 1
34. **GND** - Ground
35. **GPIO19/SPI1_MISO/PWM1** - Digital I/O 19 / SPI1 MISO / Pulse Width Modulation 1

36. **GPIO16/SPI1_CE2** - Digital I/O 16 / SPI1 CE2
37. **GPIO26** - Digital I/O 26
38. **GPIO20/SPI1_MOSI** - Digital I/O 20 / SPI1 MOSI
39. **GND** - Ground
40. **GPIO21/SPI1_SCK** - Digital I/O 21 / SPI1 SCK

The *raspi-gpio* Command Line Utility

- The *raspi-gpio* tool comes preinstalled on *Raspberry Pi OS* and can be utilized from the command line or within scripts to manipulate GPIO pins.
- Print program help.
  `$ raspi-gpio help`
- Read the attributes of all pins.
  `$ raspi-gpio get`
- Read the attributes of a single pin.
  Syntax: `$ raspi-gpio get <pin>`
  where <pin> = BCM pin number.
  Example: `$ raspi-gpio get 21`
- Set the attributes for a single pin.
  Syntax: `$ raspi-gpio set <pin> <options>`
  where <pin> = BCM pin number; <options> = combination of ip (input), op (output), dh (drive high), dl (drive low), pd (pull-down), pu (pull-up), pn (no pull), or a0-a5 (alt0-alt5), separated by spaces.
  Example (set pin 21 as a pull-up enabled input): `$ raspi-gpio set 21 ip pu`
  Example (set pin 21 as an output driven low): `$ raspi-gpio set 21 op dl`

Other Useful Commands

- Show the pinout diagram.
  `$ pinout`
- Display the I2C addresses in use.
  `$ i2cdetect -y 1`

Popular GPIO Programming Libraries

- *RPi.GPIO* - Python based GPIO interface library.
- *GPIO Zero* - Python based GPIO interface library with extensive support for many devices.
- *Blinka* - Python based library providing CircuitPython language support.
- *pigpio* - C/C++ based GPIO interface library.
- *WiringPi* - Very popular C based GPIO interface library that was recently deprecated.
- *I2Cdevlib* - C based I2C bus library with support for many I2C based devices.
- *SwiftyGPIO* - Swift based GPIO interface library.

==

Example Programs

==

Example Blink Program Using Python With The RPi.GPIO Library

```
# blink_rpigpio.py

from time import sleep
import RPi.GPIO as GPIO

RED_LED = 21

GPIO.setmode(GPIO.BCM)
GPIO.setup(RED_LED, GPIO.OUT)
print("Press CTRL-C to exit.")
try:
    while True:
        GPIO.output(RED_LED, GPIO.HIGH)
        sleep(1)
        GPIO.output(RED_LED, GPIO.LOW)
        sleep(1)
finally:
    GPIO.cleanup()
```

Run: `$ python blink_rpigpio.py`
Exit: CTRL-C

Example Blink Programs Using Python With The GPIO Zero Library

```
# blink_gpiozero_blink.py

from signal import pause
from gpiozero import LED

red_led = LED(21)

print("Press CTRL-C to exit.")
red_led.blink()
pause()
```

Run: `$ python blink_gpiozero_blink.py`
Exit: CTRL-C

```
# blink_gpiozero_loop.py

from time import sleep
from gpiozero import LED

red_led = LED(21)

print("Press CTRL-C to exit.")
while True:
    red_led.on()
    sleep(1)
    red_led.off()
    sleep(1)
```

Run: `$ python blink_gpiozero_loop.py`
Exit: CTRL-C

Example Blink Program Using CircuitPython With The Blinka Library

```
# blink_blinka.py

from time import sleep
import board
from digitalio import DigitalInOut, Direction
import RPi.GPIO as GPIO

red_led = DigitalInOut(board.D21)
red_led.direction = Direction.OUTPUT

print("Press CTRL-C to exit.")
try:
    while True:
        red_led.value = True
        sleep(1)
        red_led.value = False
        sleep(1)
finally:
    GPIO.cleanup()
```

Run: `$ python blink_blinka.py`
Exit: CTRL-C

Example Blink Program Using C With The pigpio Library

```
// blink_pigpio.c

#include <signal.h>
#include <stdio.h>
#include <pigpio.h>

const int RedLED = 21;
volatile sig_atomic_t signal_received = 0;

void sigint_handler(int signal) {
   signal_received = signal;
}

int main() {
   if (gpioInitialise() == PI_INIT_FAILED) {
      printf("ERROR: Failed to initialize the GPIO interface.\n");
      return 1;
   }
   gpioSetMode(RedLED, PI_OUTPUT);
   signal(SIGINT, sigint_handler);
   printf("Press CTRL-C to exit.\n");
   while (!signal_received) {
      gpioWrite(RedLED, PI_HIGH);
      time_sleep(1);
      gpioWrite(RedLED, PI_LOW);
      time_sleep(1);
   }
   gpioSetMode(RedLED, PI_INPUT);
   gpioTerminate();
   printf("\n");
   return 0;
}
```

Compile:	`$ gcc -Wall -o blink_pigpio blink_pigpio.c -lpigpio`
Run:	`$ sudo ./blink_pigpio`
Exit:	CTRL-C

Example Blink Program Using C++ With The pigpio Library

```
// blink_pigpio.cpp

#include <csignal>
#include <iostream>
#include <pigpio.h>

const int RedLED = 21;
volatile sig_atomic_t signal_received = 0;

void sigint_handler(int signal) {
   signal_received = signal;
}

int main() {
   if (gpioInitialise() == PI_INIT_FAILED) {
      std::cout << "ERROR: Failed to initialize the GPIO interface."
<< std::endl;
      return 1;
   }
   gpioSetMode(RedLED, PI_OUTPUT);
   signal(SIGINT, sigint_handler);
   std::cout << "Press CTRL-C to exit." << std::endl;
   while (!signal_received) {
      gpioWrite(RedLED, PI_HIGH);
      time_sleep(1);
      gpioWrite(RedLED, PI_LOW);
      time_sleep(1);
   }
   gpioSetMode(RedLED, PI_INPUT);
   gpioTerminate();
   std::cout << std::endl;
   return 0;
}
```

Compile: `$ g++ -Wall -o blink_pigpio blink_pigpio.cpp -lpigpio`
Run: `$ sudo ./blink_pigpio`
Exit: CTRL-C

==

Revisions

==

Version	Date	Modifications
1.5	12/06/2022	- Restructured cheatsheet into different sections and added additional information and useful commands. - Changed name of OS from Raspbian to Raspberry Pi OS. - Changed use of older style apt-* commands to newer and simpler apt command. - Updated example blink programs, including changing use of the C/C++ library from WiringPi to pigpio, due to WiringPi being deprecated. - Changed usage of command line utility from gpio to raspi-gpio due to WiringPi being deprecated. - Other minor changes.
1.4	04/18/2020	- Changed name of redLED constant to RedLED for C/C++ programs. - Changed name of redLED constant to RED_LED for Python (RPi.GPIO) program. - Changed name of redLED variable to red_led for Python (GPIO Zero) program. - Removed unnecessary cleanup for Python (GPIO Zero) program. - Changed namespace usage for C++ program. - Changed copyright symbol in program headers.
1.3	05/09/2019	- Changed default CLI prompts from % to $. - Added how to show package information and how to remove a package. - Updated Package.swift file for Swift 5.0 compatibility.
1.2	02/07/2019	Updated general wording and formatting for better clarification.
1.1	01/24/2019	Updated Swift example blink program to support Swift 4.
1.0	09/13/2018	Initial version.