## Parallel pings

When it comes to 255 addresses, the delay gets accumulated and becomes large. We can run all the `ping` commands in parallel to make this faster. To make the `ping` commands run in parallel, we enclose the loop body in `( )&`. `( )` encloses a block of commands to run as a subshell and `&` sends it to the background. For example:

```
#!/bin/bash
#Filename: fast_ping.sh
# Change base address 192.168.0 according to your network.

for ip in 192.168.0.{1..255} ;
do
   (
      ping $ip -c2 &> /dev/null ;

      if [ $? -eq 0 ];
      then
       echo $ip is alive
      fi
   )&
  done
wait
```

In the `for` loop, we execute many background processes and come out of the loop, terminating the script. In order to prevent the script to terminate until all its entire child processes end, we have a command called `wait`. Place `wait` at the end of the script, so that it waits for the time until all the child `( )` subshell processes complete.

### Using fping

The second method uses a different command called `fping`. It can ping a list of IP addresses simultaneously and respond very quickly. The options available with `fping` are as follows:

- The `-a` option with `fping` specifies to print all alive machine's IP addresses
- The `-u` option with `fping` specifies to print all unreachable machines
- The `-g` option specifies to generate a range of IP addresses from slash-subnet mask notation specified as IP/mask or start and end IP addresses as:

  ```
  $ fping -a 192.160.1/24 -g
  ```

  Or

  ```
  $ fping -a 192.160.1 192.168.0.255 -g
  ```

- `2>/dev/null` is used to dump error messages printed due to an unreachable host to null device

It is also possible to manually specify a list of IP addresses as command-line arguments or as a list through `stdin`. For example:

```
$ fping -a 192.168.0.1 192.168.0.5 192.168.0.6
# Passes IP address as arguments
$ fping -a < ip.list
# Passes a list of IP addresses from a file
```

## See also

- The *Playing with file descriptors and redirection* recipe of *Chapter 1*, *Shell Something Out*, explains the data redirection
- The *Comparisons and tests* recipe of *Chapter 1*, *Shell Something Out*, explains numeric comparisons

## Running commands on a remote host with SSH

SSH is an interesting system administration tool that gives you access to a shell on a remote computer which you can use to run commands. **SSH** stands for **Secure Shell** as it transfers the network data transfer over an encrypted tunnel. This recipe will introduce different ways in which commands can be executed at a remote host.

### Getting ready

SSH doesn't come preinstalled with all GNU/Linux distributions, and you may have to install the `openssh-server` and `openssh-client` packages using a package manager. SSH service runs at default port number 22.

### How to do it...

1. To connect to a remote host with the SSH server running, use:

   ```
   $ ssh username@remote_host
   ```

   In this command:

   - `username` is the user that exists at the remote host
   - `remote_host` can be the domain name or IP address