El examen consta de 20 cuestiones de opción múltiple. En cada cuestión hay una sola respuesta correcta. Las respuestas deben proporcionarse en una hoja SEPARADA que se ha facilitado junto a este enunciado.

Todas las cuestiones tienen el mismo valor. Si se responden correctamente, aportan 0.5 puntos a la nota obtenida. Si la respuesta es negativa, la aportación es negativa y equivalente a 1/5 del valor correcto; es decir, -0.1 puntos. En caso de duda se recomienda dejar la respuesta en blanco.

La duración de esta parte del examen es 1 hora.

1. La orden "`git push origin master`"

A	Trae todos los commits del depósito remoto "`origin`" a nuestro depósito local.
B	Borra el directorio de trabajo actual.
C	Siempre copia los commit locales en el depósito remoto.
D	Cuando tiene éxito, deja la rama del depósito remoto master apuntando al mismo commit que la rama master local.
E	Todas las anteriores.
F	Ninguna de las anteriores.

2. Las dos órdenes "`git pull`" y "`git fetch`"

A	Son equivalentes.
B	Traen todos los commits del depósito remoto al depósito local.
C	Dejan la rama master local apuntando al mismo commit que la rama master remota.
D	Modifican el directorio de trabajo.
E	Todas las anteriores.
F	Ninguna de las anteriores.

3. GitLab es…

A	Un servidor concreto dentro de la UPV.
B	El nombre de un depósito.
C	Una forma de confirmar (*commit*) cambios en un depósito.
D	Una versión experimental de Git.
E	Todas las anteriores.
F	Ninguna de las anteriores.

4. En el flujo de trabajo para la fase de desarrollo que se ha solicitado utilizar en GitLab...

A	Cada desarrollador del equipo debe tener un depósito remoto privado.
B	Hay un depósito canónico que cada miembro del equipo puede leer y que es diferente de los depósitos remotos privados.
C	Solo el propietario puede escribir en su depósito remoto privado.
D	Solo un miembro del equipo debe escribir en el depósito canónico.
E	Todas las anteriores.
F	Ninguna de las anteriores.

5. En un equipo TSR, cada depósito privado remoto...

A	Debe ser accesible solo por su propietario.
B	Debe ser accesible en modo escritura por su propietario.
C	Debe ser accesible en modo escritura por el integrador del equipo.
D	Debe ser accesible en modo escritura por el líder del equipo.
E	Todas las anteriores.
F	Ninguna de las anteriores.

6. Un depósito Git local…

A	Debe estar asociado a un depósito Git remoto.
B	Solo puede asociarse a un único depósito remoto.
C	Puede clonarse sobre otro depósito local.
D	No puede ser modificado.
E	Todas las anteriores.
F	Ninguna de las anteriores.

7. Seleccione la orden que crea un depósito Git local vacío...

A	“git init; touch README; git commit”
B	“git clone git:init”
C	“git init origin master”
D	“git init -u origin master”
E	Todas las anteriores.
F	Ninguna de las anteriores.

8. Asuma la siguiente situación: El depósito R (cuya URL es R_{url}) puede ser accedido por dos desarrolladores D1 y D2. La rama "master" de R apunta a un commit C que contiene el fichero "`foo.js`", con una sola línea, "`var fs = require('fs')`". Cada desarrollador ejecuta "`git clone` R_{url}`; cd R;`" en su ordenador. D1 edita "`foo.js`" y añade la línea "`fs.readFileSync('myfile')`". D2 también edita "`foo.js`" pero añade la línea "`fs.readFileSync('nofile')`". Ambos realizan el commit de sus cambios, en distintos momentos, obteniendo los commits C1 y C2, respectivamente. Entonces ellos ejecutan "`git push`". Podemos afirmar que…

A	Nada cambia en R.
B	La rama "master" de R apuntará a C1.
C	La rama "master" de R apuntará a C2.
D	Uno o más desarrolladores obtendrán un error cuando ejecuten "`git push`".
E	Todas las anteriores.
F	Ninguna de las anteriores.

Considere el siguiente fragmento 1:

```
01:  var net = require('net');
02:  var server = net.createServer(  function(c) {
03:     console.log('server connected');
04:     c.write('World');
05:     c.end();
06:  });
07:
08:  server.listen(8000, function() {
09:    console.log('server bound');
10:  });
11:
```

9. Asuma que el proceso P1 ejecuta el código del fragmento 1. Si asumimos que P2 es otro proceso iniciado en el mismo ordenador que P1 y que P2 se conecta al puerto 8000, entonces…

A	P1 envía la respuesta a P2 después de recibir el mensaje de petición.
B	Cuando P1 se pone en escucha imprime el mensaje 'server connected'.
C	P2 podría recibir el mensaje 'World' antes de enviar ninguna información a P1.
D	Si P2 cierra su socket no implica que P1 se desconecte.
E	Todas las anteriores.
F	Ninguna de las anteriores.

Considere el siguiente fragmento 2:

```
12:   var net = require('net');
13:
14:   var server_port = process.argv[2];
15:   var texto       = process.argv[3].toString();
16:   var client      = net.connect({port: parseInt(server_port}, function() {
17:      console.log('client connected');
18:      // This will be echoed by the server.
19:      client.write(texto);
20:   });
21:   client.on('data', function(data) {
22:     console.log(data.toString());
23:     client.end();
24:   });
25:
```

10. Suponga que el proceso P1 ejecuta este fragmento 2. Sea P2 un proceso servidor que escucha peticiones en el puerto 8000. Entonces …

A	P1 saca un mensaje por pantalla cuando P2 cierra el socket.
B	La conexión se producirá aunque P2 no se ejecute en la misma máquina que P1.
C	Si el servidor no está en marcha el mensaje contenido en la variable 'texto' quedará a la espera en el buffer de salida hasta que el servidor esté en marcha.
D	Se podría invocar P1 desde la línea de órdenes de Linux como : $ node client 127.0.0.1 8000 hello
E	Todas las anteriores.
F	Ninguna de las anteriores.

Considere el fragmento 3 siguiente:

```
26:   var net = require('net');
27:
28:   var LOCAL_PORT  = 8000;
29:   var LOCAL_IP   = '127.0.0.1';
30:   var REMOTE_PORT = 80;
31:   var REMOTE_IP = '158.42.156.2';
32:
33:   var server = net.createServer(function (socket) {
34:       socket.on('data', function (msg) {
35:          var serviceSocket = new net.Socket();
36:          serviceSocket.connect(parseInt(REMOTE_PORT), REMOTE_IP, function (){
37:              serviceSocket.write(msg);
38:          });
39:          serviceSocket.on('data', function (data) {
40:              socket.write(data);
41:          });
42:          console.log("Client connected");
43:       });
44:   }).listen(LOCAL_PORT, LOCAL_IP);
45:   console.log("TCP server accepting connection on port: " + LOCAL_PORT);
```

Asuma que este código se ejecuta para generar el proceso P1. Los datos de los clientes conectados al puerto 8000 se asume que llegan en bloques a través de la conexión que estos establecen. Cada bloque genera un evento "data" en el socket que mantiene la conexión.

Asuma un proceso P2 iniciado tras P1 en el ordenador de P1 y que también se conecta al puerto local 8000. Responda las tres cuestiones siguientes:

11. Cuando P2 se conecta al puerto local 8000, P1 …

A	… se conecta a REMOTE_IP.
B	… imprime '`Client connected`' en la consola.
C	… empieza a esperar datos enviados desde REMOTE_IP.
D	… deja de aceptar nuevas conexiones.
E	Todas las anteriores.
F	Ninguna de las anteriores.

12. Asumamos que P2 envía dos bloques grandes de datos en secuencia, D1 y después D2, a través de su conexión con P1.
Entonces, en ausencia de fallos,...

A	D2 siempre llega a REMOTE_IP tras D1.
B	D1 siempre llega a REMOTE_IP tras D2.
C	Solo D1 se transmite a REMOTE_IP.
D	Solo D2 se transmite a REMOTE_IP.
E	Todas las anteriores.
F	Ninguna de las anteriores.

13. Asumamos que el servidor en REMOTE_IP solo pudiera manejar una conexión a la vez.
Entonces, si P2 enviase dos bloques de datos...

A	El servidor obtendría dos bloques de datos.
B	El servidor terminaría tras recibir el primer bloque de datos.
C	El servidor recibe un solo bloque de datos.
D	El proxy se cierra tras manejar el primer bloque de datos.
E	Todas las anteriores.
F	Ninguna de las anteriores.

Considere el fragmento 4 siguiente:

```
01:   var net = require('net');
02:
03:   var LOCAL_PORT  = 8000;
04:   var LOCAL_IP  = '127.0.0.1';
05:   var REMOTE_PORT = 80;
06:   var REMOTE_IP = '158.42.156.2';
07:
08:   var server = net.createServer(function (socket) {
09:       console.log("Client connected");
```

```
10:       var serviceSocket = new net.Socket();
11:       serviceSocket.connect(parseInt(REMOTE_PORT), REMOTE_IP, function (){
12:          socket.on('data', function (msg) {
13:              serviceSocket.write(msg);
14:          });
15:          serviceSocket.on('data', function (data) {
16:              socket.write(data);
17:          });
18:       });
19:   }).listen(LOCAL_PORT, LOCAL_IP);
20:   console.log("TCP server accepting connection on port: " + LOCAL_PORT);
```

Asuma que P1 ejecuta este código en vez del fragmento 3, siendo el resto idéntico a lo descrito para el fragmento 3. Conteste las tres cuestiones siguientes:

14. Cuando P2 se conecta al puerto local 8000, P1 …

A	… se conecta a REMOTE_IP
B	… imprime '`Client connected`' en la consola.
C	… crea como máximo una conexión a REMOTE_IP por cada conexión cliente.
D	… espera datos desde REMOTE_IP tan pronto como establezca una conexión con REMOTE_IP.
E	Todas las anteriores.
F	Ninguna de las anteriores.

15. P2 envía dos bloques grandes de datos en secuencia, D1 seguido por D2, a través de su conexión con P1.
Entonces, en ausencia de fallos …

A	D2 siempre llega a REMOTE_IP tras D1.
B	D1 siempre llega a REMOTE_IP tras D2.
C	Solo D1 es transmitido a REMOTE_IP.
D	Solo D2 es transmitido a REMOTE_IP.
E	Todas las anteriores.
F	Ninguna de las anteriores.

16. Asuma que el servidor en REMOTE_IP pueda gestionar solo una conexión a la vez. Entonces, si P2 envía dos bloques de datos …

A	El servidor obtiene ambos bloques de datos.
B	El servidor termina tras recibir el primer bloque de datos.
C	El servidor solo recibe un bloque de datos.
D	El proxy se cierra tras manejar el primer bloque de datos.
E	Todas las anteriores.
F	Ninguna de las anteriores.

17. Ambos fragmentos 3 y 4 …

A	Permiten a P1 cerrar la conexión a REMOTE_IP cuando el cliente cierra la conexión.
B	Permiten a P1 cerrar la conexión con sus clientes cuando REMOTE_IP cierre su conexión.
C	Necesitan ser modificados para capturar el evento 'end' en `serviceSocket` y `socket` para gestionar adecuadamente el cierre de conexiones.
D	No pueden ser modificados para gestionar adecuadamente el cierre de conexiones.
E	Todas las anteriores.
F	Ninguna de las anteriores.

Considere el siguiente fragmento 5:

```
01:  var net = require('net');
02:
03:  var COMMAND_PORT  = 8000;
04:  var LOCAL_IP  = '127.0.0.1';
05:  var BASEPORT  = COMMAND_PORT + 1;
06:  var remotes = []
07:
08:  function Remote(host, port, localPort) {
09:      this.targetHost = host;
10:      this.targetPort = port;
11:      this.server = net.createServer(function (socket)
12:          WHO.handleClientConnection(socket);
13:      });
14:      if (localPort) {
15:          this.server.listen(localPort, LOCAL_IP);
16:      }
17:  }
18:
19:  Remote.prototype.handleClientConnection = function (socket) {
20:      var serviceSocket = new net.Socket();
21:      serviceSocket.connect(X_SERVER_PORT, X_SERVER_IP, function (){
22:         socket.on('data', function (msg) {
23:             serviceSocket.write(msg);
24:         });
25:         serviceSocket.on('data', function (data) {
26:             socket.write(data);
27:         });
28:         X_WHAT.on(X_EVENT, function () {
29:             serviceSocket.end();
30:         });
31:         X_WHICH.on(X_EVENT, function () {
32:             WHAT.end();
33:         });
34:      });
35:  }
36:
37:  Remote.prototype.start = function (localPort) {
38:     this.server.listen(localPort, LOCAL_IP);
39:  }
40:
```

18. **En el fragmento 5, la variable** `WHO`

A	Debe declararse en el ámbito de la función `Remote` .
B	Debe apuntar al objeto que se está construyendo.
C	No puede ser null .
D	No puede ser undefined.
E	Todas las anteriores.
F	Ninguna de las anteriores.

19. **En el fragmento 5,** `X_WHAT` **debe sustituirse por...**

A	El socket de la conexión cliente.
B	El socket de la conexión del servidor.
C	El objeto Remote.
D	El objeto servidor.
E	Todas las anteriores.
F	Ninguna de las anteriores.

20. **En el fragmento 5,** `X_SERVER_PORT` **y** `X_SERVER_IP` **...**

A	Se refieren a los atributos `targetHost` y `targetPort` del objeto Remote.
B	Deben sustituirse por `this.targetHost` y `this.targetPort`, respectivamente.
C	Pueden ser idénticos para diferentes objetos Remote.
D	No pueden ser null o undefined
E	Todas las anteriores.
F	Ninguna de las anteriores.