```
/*
 * The FTPimp testing suite
 * (c) 2014 Nicholas Riley, Sparkida. All Rights Reserved.
 * @module test/index
 */
var assert = require('assert'),
        fs = require('fs'),
        FTP = require('../'),
        Queue = FTP.prototype.Queue,
        config = require('../config'), // jshint ignore:line
        path = require('path'),
        ftp;
config.debug = false;
describe('FTPimp', function () {
        //TODO - change to main
        before(function (done) {
                this.timeout(10000);
                /**create new FTP instance connection
                 * and login are automated */
                ftp = FTP.create(config, false);
                ftp.connect(done);
        });

        describe('Simple commands have a "raw" property string of the command',
function () {
                var com = {
                                ls: 'LIST',
                                lsnames: 'NLST',
                                port: 'PORT',
                                pasv: 'PASV',
                                chdir: 'CWD',
                                mkdir: 'MKD',
                                rmdir: 'RMD',
                                type: 'TYPE',
                                rename: 'RNTO',
                                get: 'RETR',
                                filemtime: 'MDTM',
                                unlink: 'DELE',
                                getcwd: 'PWD',
                                ping: 'NOOP',
                                stat: 'STAT',
                                info: 'SYST',
                                abort: 'ABOR',
                                quit: 'QUIT'
                        };
                Object.keys(com).forEach(function (key) {
                        it('FTP.prototype.' + key + ' has raw ' + com[key], function
() {
                                assert.equal(FTP.prototype[key].raw, com[key]);
                        });
                });
        });

        var testDir = 'ftpimp.test.' + String(new Date().getTime()).slice(3) +
'.tmp';
        describe('mkdir#MKD: make a remote directory', function () {
                it ('succeeds', function (done) {
                        ftp.mkdir(path.join(testDir, 'foo'), function (err, res) {
                                assert(res.length, 2, 'Could not add directories');
                                done(err);
                        }, true);
                });
                it ('succeeds at making a directory with a space', function (done) {
                        ftp.mkdir(path.join(testDir, 'foo bar'), function (err, res)
{
                                assert(res.length, 1, 'Could not add directories');
                                done(err);
                        }, true);
                });
                it ('fails', function (done) {
                        ftp.mkdir('', function (err, res) {
                                assert(err instanceof Error);
                                assert(!res);
                                done();
                        });
                });
        });

        describe('chdir#CWD: change working directory', function () {
                it ('fails', function (done) {
                        ftp.chdir('somebadlookup', function (err, res) {
                                assert(err instanceof Error);
                                assert(!res);
                                done();
                        });
                });
                it ('succeeds, changing to testDir - ' + testDir, function (done) {
                        ftp.chdir(testDir, function (err, res) {//testDir, function
(err, res) {
                                assert(typeof res === 'string');
                                done(err);
                        });
                });
        });

        describe('type#TYPE: set transfer types', function () {
                it ('fails', function (done) {
                        ftp.type('badTypeError', function (err, res) {
                                assert(err instanceof Error);
                                assert(!res);
                                done();
                        });
                });
                it ('changed type to image (binary data)', function (done) {
                        ftp.type('binary', function (err, res) {
                                assert(res);
                                done(err);
                        });
                });
                it.skip ('changed type to EBCDIC text(not available)', function
(done) {
                        ftp.type('ebcdic', function (err, res) {
                                assert(res);
                                done(err);
                        });
                });
                it ('changed type to local format', function (done) {
                        ftp.type('local', function (err, res) {
                                assert(res);
                                done(err);
                        });
                });
                it ('changed type to ASCII text', function (done) {
                        ftp.type('ascii', function (err, res) {
                                assert(res);
                                done(err);
                });
        });
    });

    describe('setType: set transfer type based on file', function () {
        it ('uses ASCII as default', function (done) {
            ftp.setType('badTypeError', function (err, res) {
                assert(ftp.currentType, 'ascii');
                done();
            });
        });
        it ('changes to binary for images', function (done) {
            ftp.setType('test.png', function (err, res) {
                assert(ftp.currentType, 'binary');
                done();
            });
        });
        it ('changes to ASCII for text', function (done) {
            ftp.setType('index.js', function (err, res) {
                assert(ftp.currentType, 'ascii');
                done();
            });
        });
    });

    describe('put: transfers files to remote', function () {
        it('succeeds', function (done) {
            ftp.put(['./test/index.js', 'index.js'], function (err, res)
{
                assert.equal(res, 'index.js');
                ftp.put(['./test/test.png', 'test.png'], function
(err, res) {
                    assert.equal(res, 'test.png');
                    done(err);
                });
            });
        });
        it ('fails', function (done) {
            ftp.put('badFileError', function (err, res) {
                assert(err instanceof Error);
                assert(!res);
                done();
            });
        });
    });

    describe('rename#RNTO: rename to', function () {
        it ('succeeds', function (done) {
            ftp.rename(['index.js', 'ind.js'], function (err, res) {
                assert(!!res);
                done(err);
            });
        });
        it ('fails', function (done) {
            ftp.rename(['missingFile', 'foo'], function (err, res) {
                assert(err instanceof Error);
                assert(!res);
                done();
            });
        });
    });

    describe('get#RETR: retrieve remote file', function () {
```

```
		it ('succeeds at getting ASCII text file', function (done) {
			ftp.get('ind.js', function (err, res) {
				assert(typeof res === 'string');
				done(err);
			});
		});
		it ('succeeds at getting binary image file', function (done) {
			ftp.type('binary', function (err, res) {
				if(err) {
					done(err);
				} else {
					ftp.get('test.png', function (err, res) {
						if (err) {
							done(err);
						} else {
							assert(typeof res === 
'string');
							done();
						}
					});
				}
			});

		});
		it ('fails', function (done) {
			ftp.get('fileNotFoundError', function (err, res) {
				assert(err instanceof Error);
				assert(!res);
				done();
			});
		});
	});

	describe('save: retrieve remote file and save to local', function () {
		var saved = [];
		it.only ('succeeds', function (done) {
			ftp.save(['./foo/foo.pdf', './test/fi.pdf'], function (err, 
res) {
				assert(!!res);
				saved.push(res);
				done();
				//fs.unlink('ind-ftpimp-remote-saved.js', function 
(delError, res) {});
			});
			/*
			ftp.save(['ind.js', './test/ind-ftpimp-remote-saved2.js'], 
function (err, res) {
				assert(!!res);
				saved.push(res);
				//assert.deepEqual(['ind-ftpimp-remote-saved.js', 
'ind-ftpimp-remote-saved2.js'], saved);
				//fs.unlink('ind-ftpimp-remote-saved2.js', function 
(delError, res) {
				//	done(delError);
				//});
				done();
			});*/
		});
		it ('fails', function (done) {
			ftp.save(['missingFile', 'foo'], function (err, res) {
				assert(err instanceof Error);
				assert(!res);
				done();
				});
			});
		});

		describe('filemtime#MDTM: return the modification time of a remote file',
function () {
			it ('succeeds', function (done) {
				ftp.filemtime('ind.js', function (err, res) {
					assert(!isNaN(Number(res)));
					done(err);
				});
			});
			it ('fails', function (done) {
				ftp.filemtime('fileNotFoundError', function (err, res) {
					assert(err instanceof Error);
					assert(!res);
					done();
				});
			});
		});

		describe('ls#LIST: list remote files', function () {
			it ('succeeds', function (done) {
				ftp.ls('', function (err, res) {
					assert(Array.isArray(res));
					done(err);
				});
			});
			it ('fails', function (done) {
				ftp.ls('somebadlookup', function (err, res) {
					assert(!err);
					assert(Array.isArray(res), 'expected array result');
					assert.equal(res.length, 0);
					done();
				});
			});
		});

		describe('lsnames#NLST: name list of remote directory', function () {
			it ('succeeds', function (done) {
				ftp.lsnames('', function (err, res) {
					assert(Array.isArray(res));
					done(err);
				});
			});
			it ('fails', function (done) {
				ftp.lsnames('somebadlookup', function (err, res) {
					assert(!err);
					assert(Array.isArray(res), 'expected array result');
					assert.equal(res.length, 0);
					done();
				});
			});
		});

		describe('unlink#DELE: delete remote file', function () {
			it ('succeeds', function (done) {
				ftp.unlink('ind.js', function (err, res) {
					assert.equal(res, 'ind.js');
					ftp.unlink('test.png', function (err, res) {
						assert.equal(res, 'test.png');
						done(err);
					});
			});
		});
		it ('fails', function (done) {
			ftp.unlink('fileNotFoundError', function (err, res) {
				assert(err instanceof Error);
				assert(!res);
				done();
			});
		});
	});

	describe('root: changes to root directory', function () {
		it ('succeeds', function (done) {
			ftp.root(function (err, res) {
				assert(typeof res === 'string');
				done(err);
			});
		});
	});

	describe('rmdir#RMD: recursively remove remote directory', function () {
		this.timeout(10000);
		it ('should recursively remove the directory ' + testDir, function 
(done) {
			ftp.mkdir(path.join(testDir, 'foo'), function(){}, true);
			ftp.rmdir(testDir, function (err, res) {
				assert(!err, err);
				assert.equal(res.length, 3);
				done();
			}, true);
		});
		it ('should remove the directory even if it is the only object to be 
removed', function (done) {
			ftp.mkdir(testDir, function(){}, true);
			ftp.rmdir(testDir, function (err, res) {
				assert(!err, err);
				assert.equal(res.length, 1);
				done();
			}, true);
		});
		it ('should recursively remove the directory ' + testDir, function 
(done) {
			ftp.mkdir(path.join(testDir, 'foo'), function(){}, true);
			ftp.rmdir(testDir, function (err, res) {
				assert(!err, err);
				assert.equal(res.length, 2);
				done();
			}, true);
		});
		it ('should recursively remove the directory in queue order: ' + 
testDir, function (done) {
			ftp.mkdir(path.join(testDir, 'foo'), function(){
				ftp.put(['./test/test.png', path.join(testDir, 
'test.png')], function(){}, Queue.RunNext);
			}, true);
			ftp.rmdir(testDir, function (err, res) {
				assert(!err, err);
				assert.equal(res.length, 3);
				done();
			}, true);
		});
		it ('should recursively remove files in the directory ' + testDir, 
function (done) {
			ftp.mkdir(path.join(testDir, 'foo'), function(){
				ftp.put(['./test/test.png', path.join(testDir, 
'foo.png')], function(){}, Queue.RunNext);
				ftp.put(['./test/test.png', path.join(testDir, 
'foo1.png')], function(){}, Queue.RunNext);
				ftp.put(['./test/test.png', path.join(testDir, 
'foo2.png')], function(){}, Queue.RunNext);
			}, true);

			ftp.rmdir(testDir, function (err, res) {
				assert(!err, err);
				assert.equal(res.length, 5);
				done();
			}, true);
		});
		it ('should recursively remove all empty directories', function 
(done) {
			ftp.mkdir(path.join(testDir, 'foo', 'bar', 'who'), 
function(){
			}, true);

			ftp.rmdir(testDir, function (err, res) {
				assert(!err, err);
				assert.equal(res.length, 4);
				done();
			}, true);
		});
		it ('should recursively remove files in the directory ' + testDir, 
function (done) {
			ftp.mkdir(path.join(testDir, 'foo'), function(){
				ftp.put(['./test/test.png', path.join(testDir, 
'foo', 'foo.png')], function(){}, Queue.RunNext);
				ftp.put(['./test/test.png', path.join(testDir, 
'foo', 'foo1.png')], function(){}, Queue.RunNext);
				ftp.put(['./test/test.png', path.join(testDir, 
'test.png')], function(){}, Queue.RunNext);
				ftp.put(['./test/test.png', path.join(testDir, 
'test1.png')], function(){}, Queue.RunNext);
			}, true);

			ftp.rmdir(testDir, function (err, res) {
				assert(!err, err);
				assert.equal(res.length, 6);
				done();
			}, true);
		});
		it ('fails', function (done) {
			ftp.rmdir('badDirectoryError', function (err, res) {
				assert(err instanceof Error);
				assert(!res);
				done();
			}, true);
		});
	});

	describe('General FTP commands', function () {
		it ('ping#NOOP: do nothing, ping the remote server', function (done) 
{
			ftp.ping(done);
		});
		it ('stat#STAT: get server status', function (done) {
			ftp.stat(function (err, res) {
				assert(typeof res === 'string');
```

```
                        done(err);
                });
        });
        it ('getcwd#PWD: gets current working directory', function (done) {
                ftp.getcwd(function (err, res) {
                        assert(typeof res === 'string');
                        done(err);
                });
        });
        it ('info#SYST: return system type', function (done) {
                ftp.info(function (err, res) {
                        assert(typeof res === 'string');
                        done(err);
                });
        });
});

describe('Queue RunLevel Sequencing', function () {
        var order = [];
        it('should run in the order of 1,3,2,4', function (done) {
                ftp.ls('foo-1', function (err, res) {
                        order.push(1);
                });
                ftp.ls('foo-2', function (err, res) {
                        order.push(2);
                });
                ftp.ls('foo-3', function (err, res) {
                        order.push(3);
                }, Queue.RunNext);
                ftp.ls('foo-4', function (err, res) {
                        order.push(4);
                        assert.deepEqual(order, [1,3,2,4]);
                        done();
                });
        });
});

describe('Queue sequence tests', function () {
        var level = 0,
                msg = 'should be at level ';
        it ('should run in waterfall', function (done) {
                ftp.ping(function (err, res) {
                        level += 1;
                        //console.log(level);
                        assert(level, 1, msg + level);
                        ftp.runNow(ftp.ping.raw, function (err, res) {
                                level += 1;
                                //console.log(level);
                                assert(level, 2, msg + level);
                        });
                        ftp.ping(function (err, res) {
                                level += 1;
                                assert(level, 6, msg + level);
                        });
                        ftp.runNext(ftp.ping.raw, function (err, res) {
                                level += 1;
                                //console.log(level);
                                assert(level, 3, msg + level);
                        });
                });
                ftp.ping(function (err, res) {
                        level += 1;
                        //console.log(level);
```

```
                                assert.equal(level, 4, msg + level);
                                ftp.ping(function (err, res)  {
                                        level += 1;
                                        //console.log(level);
                                        assert.equal(level, 7, msg + level);
                                        done();
                                });
                        });
                        ftp.ping(function (err, res) {
                                level += 1;
                                //console.log(level);
                                assert.equal(level, 5, msg + level);
                        });
                });
                it ('should never run the last command', function (done) {
                        ftp.ping(function (err, res) {
                                level = 1;
                                assert(level, 1, msg + level);
                                setTimeout(function () {
                                        done();
                                }, 250);
                        }, Queue.RunLast, true);
                        ftp.ping(function (err, res) {
                                done('This should not run!');
                        });
                });
                it ('should override the holdQueue from last command and quit',
function (done) {
                        ftp.quit(done, Queue.RunNow);
                });
        });
});
```