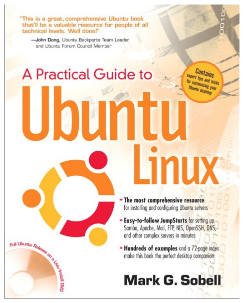
# CIS 214
# Unix and Linux Fundamentals



## Supplemental Material

An Essentials based Introduction to Unix and Linux Client Networking Applications

### Introducing Client-Side Networking

The material in the textbook has more to do with networking concepts in general and less to do with the client-side utilities for leveraging networking into productivity tool-chains. In this supplementary material to the textbook we will not cover general networking theory or practice. Neither will we cover, in any great detail, networking services, configuration or support. Rather, we will concentrate on the client-side networking utilities and protocols that support user productivity. A limited amount of this information is in the chapter 10 section on Trusted Hosts (pages 391-3). In our course, we will only be concerned with the tools that extend our shell productivity to networked hosts as described here.

*General networking theory and practice for CS transfer students and those in other related programs is covered in our CIS 70A, B and D courses.*

For our purposes, we want to understand the following two categories of client-side networking tools available on modern Linux platforms:

1. The traditional Unix remote or R-utilities (rlogin, rsh, rcp) and their secure shell counterparts the S-utilities (ssh, scp).

2. The comparable, cross-platform utilities from the application layer of the TCP stack, or TCP/IP protocol suite (telnet, ftp).

### The TCP Applications:

The TCP applications are part of the TCP/IP protocol suite, first implemented on the DoD (Department of Defense) networking model, and then made most popular, generally, with the growth of the Internet. These CLI (Command-Line Interface) applications open interactive console sessions to accomplish their tasks for remote logins and file transfers.

Although initially developed along with the Internet on Unix systems, these are cross-platform applications and therefore make no assumptions about user authentication or data translation.

For background, you should note the command-line syntax, purpose and operation of these applications.

## Telnet - TELephone NETwork (historical meaning)

*Telent is used for logins to a remote host. It authenticates to, and opens a remote shell on, another host on a network...*

```
telnet [-l UserID] HostName [PortNumber]
```

- Without supplying the user ID as an option to the command-line, you will be prompted for the user name on the remote system. In any case, you are prompted for the password.
- While the default port number is 23, supplying another port number can be useful for communicating with (or debugging) another TCP service, like an SMTP mail server.
- If successful, your keyboard and monitor are attached to a shell on the remote host.
- Now, telnet is no longer considered a secure protocol for communications over public or otherwise insecure networking infrastructures because communications are in clear-text.

## FTP - File Transfer Protocol

*FTP is used for file transfers to and from a remote host. It authenticates to, and transfers files to and from, another host on a network. If necessary, FTP can translate files during transfer...*

```
ftp [UserID@]HostName
```

- Without supplying the user ID in the optional command-line syntax, you will be prompted for the user name on the remote system. In any case, you are prompted for the password.
- Some servers allow anonymous transfers. In this case, the user name you supply should be "`anonymous`" and the password should be your `email@address` (by generally accepted Internet convention and cyberspace etiquette).
- The following, most important, commands are recognized by the FTP interactive session:

`ascii` / `binary`

- Changes the mode of transfer to *translate* ASCII files or *copy* BINARY images. This may be necessary as plain-text files require line-termination conversion to remain compatible between differing platforms while binary files must not be altered in any way.

`get FileName` / `put FileName`
`mget FileNamePattern` / `mput FileNamePattern`

- These download or upload the specified file (or files matching the file name pattern).
- Some shell commands will work in a limited capacity during an FTP session (like ls to list the contents of the logged remote directory or cd to change to a different remote directory).

## The Unix Utilities:

The Unix remote or R-utilities and their secure shell counterparts the S-utilities traditionally only work between networked Unix hosts. Therefore, these services can be configured to trust certain hosts and make assumptions about their authentication environments based on the common Unix OS services.

The R-utilities are no longer considered to be secure for use over public networks because their communications are in clear-text or otherwise unencrypted.  Because of this, modern distributions of Linux usually do not install the R-utility services (by default).  When they are installed, the init scripts are not configured to run their daemons automatically, and their service configurations are defaulted to refuse connections.  You can edit these start-up scripts and configuration files directly to change these behaviors, or use the CLI, CUI or GUI administrative tools provided by your distribution's vendor.

The S-utilities are considered to be secure because communications are wrapped in a secure shell using strong public-key encryption technologies, and as such, are usually installed and configured by default with the installation scripts provided my most modern Linux distribution Vendors.

For background, you should note the command-line syntax, purpose and operation of the R-utilities, but you will need to understand the listed, important parts of the command-line syntax, purpose and operation of the S-utilities.  The ssh utility has become so important, in modern networking, that it is now implemented as a preferred or required protocol across all other modern OS platforms.

## rlogin - Remote Login

*rlogin is used for logins to a remote host.  It authenticates to, and opens a remote shell on another host on a network...*

```
rlogin [-1 UserID] HostName
```

- Without supplying the user ID as an option to the command-line, rlogin assumes your user ID on the remote host matches your current login ID on the local host - if trusted, you are not prompted for your password.
- When supplying a remote user ID that is different from your local login, you will be prompted for the password on the remote host.
- If successful, your keyboard and monitor are attached to a shell on the remote host.

## rsh - Remote SHell

*rsh is used to run commands in a shell on a remote host and send any output back to the initiating shell on the local host.  It can also be used to open a session on a remote host.  Like rlogin, it authenticates to, and opens a remote shell on another host on a network...*

```
rsh [-1 UserID] HostName [command—line]
```

- Without supplying the user ID as an option to the command-line, rsh assumes your user ID on the remote host matches your current login ID on the local host - if trusted, you are not prompted for your password.
- When supplying a remote user ID that is different from your local login, you will be prompted for the password on the remote host.
- Without supplying the optional command sequence to execute on the remote host, rsh degenerates to rlogin; otherwise, the remote shell is closed after that command-line terminates.
- When the optional command-line syntax to execute on the remote host is supplied, the sequence is executed in a remote shell on the remote host and its standard output is transferred back to the local shell on the local host – thus, rsh command-lines can establish pipes between hosts, allowing distributed processing.

## rcp - Remote CoPy

*rcp is used to copy files from and to an account on a remote host. Remote file paths are treated as being relative to the home directory of the owner of the account on the remote host...*

```
rcp [UserID@]Host:FileName FileName                  # downloads
rcp [–r] [UserID@]Host:FilePattern Directory

rcp FileName [UserID@]Host:FileName                  # uploads
rcp [-r] FilePattern [UserID@]Host:Directory
```

- Without supplying the user ID in the optional command-line syntax, rcp assumes your user ID on the remote host matches your current login ID on the local host - if trusted, you are not prompted for your password.
- When supplying a remote user ID that is different from your local login, you will be prompted for the password on the remote host.
  - *Some implementations may not provide for a remote user ID that doesn't match the local login - in that case, the indicated syntax is not allowed and you cannot establish a connection to the remote host.*
- The source(s) is (are) the first non-option token(s) on the command-line after the utility name, the target (or destination) is the last token on the command-line.
- When the -r option is used, any source directories provided or established from wildcard patterns are recursed for transfer.

## ssh - Secure SHell

*ssh is used to run commands in a secure shell on a remote host and send any output back to the initiating shell on the local host. It can also be used to open a session on a remote host. Communications across the network are secured with strong public key cryptography.*

```
ssh [UserID@]HostName [command-line]
```

- Without supplying the user ID in the optional command-line syntax, ssh assumes your user ID on the remote host matches your current login ID on the local host - if trusted, you are not prompted for your password.
- When supplying a remote user ID that is different from your local login, you will be prompted for the password on the remote host.
- Without supplying the optional command sequence to execute on the remote host, ssh starts an interactive secure shell on the remote host; otherwise, the remote shell is closed after the command-line terminates.
- When the optional command-line syntax to execute on the remote host is supplied, the sequence is executed in a secure remote shell on the remote host and its standard output is transferred back to the local shell on the local host – therefore, ssh allows for not only pipes between utilities on a single host, as any local shell does, but for pipes between utilities on different networked hosts, establishing distributed processing capabilities right at the command-line for any normal user without the need for specialized applications or programming abilities.

As noted earlier, ssh is now implemented across all modern platforms and is an important secure networking protocol. It allows secure remote logins (if not always the advanced capabilities).

**scp - Secure CoPy**

*scp is used to copy files from and to an account on a remote host. Remote file paths are treated as being relative to the home directory of the owner of the account on the remote host. Communications across the network are secured with public key cryptography.*

```
scp [[UserID@]Host1:]File [[UserID@]Host2:]File
scp [–r] [[UserID@]Host1:]Pattern [[User@]Host2:]Directory
```

- Without supplying the user ID in the optional command-line syntax, scp assumes your user ID on any remote host matches your current login ID on the local host - if trusted, you are not prompted for your password.
- When supplying any remote user ID that is different from your local login, you will be prompted for the password(s) on the remote host(s).
- The source(s) is (are) the first non-option token(s) on the command-line after the utility name, the target (or destination) is the last token on the command-line.
- When the -r option is used, any source directories provided or established from wildcard patterns are recursed for transfer.
- Unlike rcp, scp can be used to copy files between remote hosts.

## One more very special utility: rsync

Like ssh, rsync is such an amazingly useful, network-capable utility that it has been implemented on all modern platforms, although it's usually made available on non-Unix platforms as an embedded feature of other, paid services or at additional cost, though sometimes as a free download from third parties.

Rsync can be used to copy or backup all or portions of filesystems on the same host or between networked hosts. It can work with directly connected or remote, networked filesystems.

Of all the features of rsync, the most striking may be its ability to very *quickly* copy *large* amounts of data through *limited* bandwidth connections. This is possible because rsync only needs to actually transfer the changed parts of any dataset. Therefore, for example, you might be able to update a backup of your 64GB Tablet's internal storage via a Wireless-B connection in mere seconds! That's because, under usual circumstances, only a relatively small amount of changes are made, even to large data collections, between backup intervals.

**rsync - Remote Sync**

*rsync is a fast, versatile, remote (and local) file-copying tool. When used across networks, it authenticates to remote shells on other hosts. For secure communications, rsync will use ssh...*

```
rsync [Option]... Source... [[UserID@]Host:]Destination
rsync [Option]... [[UserID@]Host:]Source... Destination
```

- The source(s) and destinations are typically directories that get recursed; however, the use of wildcards can work in surprising ways, so check with the -n option before committing:
- The syntax can get complicated to account for any imaginable situation or desired result; however, the most useful, simple options – that can accomplish most needs – are as follows:

**traditional options**:

| | |
|---|---|
| `-n` | don't do anything – just display what would happen |
| `-a` | archive mode – save original metadata including permissions and time-stamps |
| `-r` | recurse directories – included by default with -a |
| `-v` | increase the detail of the output displayed on the console's monitor |

**long options**:

| | |
|---|---|
| `--progress` | display the progress (useful for large file transfers) |
| `--delete` | delete files in the target path that aren't in the source path(s) |
| `--existing` | don't copy new files that aren't already in the target |
| `--update` | don't copy existing files that are newer in the target |

**another special long option**:

```
--exclude 'pattern'
```

The above may prove a little different from what you have learned before as the “pattern” does not have to include either wildcards (as in BASH) nor be a more advanced regular expression (as with sed or awk). It will match the specified character-string in the evaluated source's path, but in this case, order counts: You should place the exclude patterns after the include specifications that might contain them. Specifying them before the source(s) won't exclude anything.

**another consideration**:

How you specify the source(s) may also work differently than expected from your previous experience with BASH:

| | |
|---|---|
| `directory` | will copy the directory and its contents into the destination |
| `directory/` | will copy all the contents of the directory (not the directory itself) |
| `directory/*` | will copy all the visible contents of the directory (not the directory itself) |

**An example**:

```
rsync -a -v --delete --progress ~ /media/UsbPath
```

From BASH (that recognizes the special meaning of ~ in the filesystem) will copy all the contents of your home directory to the mounted USB drive path.

**Special example**:

```
rsync -rltgoD -v –delete --progress ~ /media/UsbPath
```

Look it up: But, if your attached USB device happens to be an NTFS formatted filesystem, you may get messages because the metadata for the Unix files won't match the NTFS capabilities. This is mostly the equivalent of the previous example but should avoid the warning messages.

END