

# UUCP in 2022

v20220822





Written by Lawrence Candilas



Above: This is not the UUCP we are talking about

# UUCP in 2022

## Table of Contents

UUCP in 2022 .....	2
What Is UUCP?.....	4
What does UUCP do?.....	4
UUCP's upbringing.....	4
Taylor-UUCP.....	4
UUCP network and service model .....	6
First: Two or more <i>nodes</i> .....	6
Second: Nodes need to have <i>access</i> between each other .....	6
Third: All the action happens during a <i>call</i> :.....	6
Hol up, THE OTHER SIDE CAN EXECUTE COMMANDS ON MY SYSTEM? .....	7
Fourth: <i>Systems periodically call each other</i> .....	7
UUCP access model for nodes.....	8
UUCP is really the <i>uucico</i> command – on both sides .....	8
Just in case you're really going to use a modem .....	8
Preferred access method: SSH  .....	8
Serial port (modem) login  .....	9
I was thinking of trying to use the TLI protocol...   .....	9
UUCP Access And File Security.....	10
Nodes Must “Know” Each Other Beforehand .....	10
SSH Key Files Prevent Random People From Trying To Guess names .....	10
<i>uucico</i> Doesn't Run As Root .....	10
<i>uucico</i> Has It's Own Username And Password File At <i>/etc/uucp/passwd</i> .....	11
UUCP Received/Sent Local File Permissions Are “free for all”, Though .....	11
How To Use UUCP .....	13
UUCP Addresses .....	13
Simple And Usual Example .....	13
Via Intermediate - Forwarding .....	13
Local Permissions In Order To Use UUCP Commands.....	14
<i>uuto</i> - Sending A File.....	14
<i>uulog</i> - Did It Work?.....	14
Receiving Files.....	15
<i>uupoll</i> – Call Now .....	15
Installing and Configuring Your UUCP .....	16
Receiver And Sender Roles.....	16
Preparing for either role .....	16
1. Install UUCP If You Need To (Assuming Debian).....	16
2. Read This And Understand How SSH Keys Work.....	16
3. Decide A Few Things .....	17
SSH RECEIVER role setup step-by-step.....	17
1. Create A Local User Just For Incoming UUCP Requests.....	17
2. Create SSH Key Files .....	17
A Note On Key Hygiene .....	17
3. Putting The Key In The Right Spot For <i>sshd</i> .....	18
4. Setting Up <i>.ssh/authorized_keys</i> .....	18

5. Disable Password Authentication For Your UUCP Dedicated User.....	18
6. Define Some UUCP Users And Tell UUCP Your Chosen nodename .....	18
7. Add Systems That Will Call You To /etc/uucp/sys .....	19
SSH CALLER role setup step-by-step.....	19
1. Get That Private Key File From The Receiver. ....	19
1a. Ask The Receiver To Add Your nodename To His/Her /etc/uucp/sys File .....	19
2. Put The Receiver's Private Key File In A Place <i>uucico</i> Can See And Read It.....	19
3. Edit Some UUCP Configuration Files.....	20
/etc/uucp/sys.....	20
/etc/uucp/Port .....	20
4. Log Into The Receiver Once Manually With The SSH Command In The Port File.....	20
5. The Receiver Doesn't Need Something In /etc/uucp/Port For My System, As A Caller?.....	21
Additional UUCP stuff.....	22
UUCP Node Forwarding .....	22
UUCP mail setup.....	23
Traditional UNIX Mail Workflow (Where Email Was Born) .....	23
It's Sounding Like We Gotta Setup <i>postfix</i> .....	24
Alright, Let's Do This: Configuring Postfix For UUCP .....	24
1. master.cf.....	24
2. main.cf .....	25
3. Make An /etc/postfix/transport File .....	25
4. Postfix <i>Actually Wants Its Transport File In A Database Format</i> . ....	26
5. What About The Other Side?.....	27
6. Testing.....	28
7. Setup Errors You May Run Into And Fix Suggestions .....	28
External Email Clients.....	29
Serial port-based nodes .....	30
No Keys With Serial .....	30
Serial RECEIVER role setup step-by-step.....	30
1. Create a user just for incoming serial UUCP requests (separate from SSH-based user!) .....	30
2. Set login shell to <i>uucico</i> .....	30
3. Give incoming serial UUCP user a really strong password .....	30
4. Tell <i>systemd</i> to maintain a <i>getty</i> on /dev/ttyS0 .....	31
5. Add the system that will call you through serial port to /etc/uucp/sys .....	31
Serial CALLER role setup step-by-step.....	31
1. Receiver's nodename must appear the /etc/uucp/sys file. ....	31
2. Edit some UUCP configuration files .....	32
/etc/uucp/sys.....	32
/etc/uucp/Port .....	32
Configuration References .....	33

# What Is UUCP?

- UUCP stands for UNIX to UNIX CoPy.
- You don't have to be a Unitarian Universalist Congregation of Phoenix member to use it (/s).
- It's old as hell, but not quite as old as that church.



## What does UUCP do?

- It can be used to send/receive files, emails, and news.
  - Yes, you can already do that but UUCP doesn't need TCP/IP to do it unlike your fancy IP-based services! (But you'll probably use it over TCP/IP and SSH in real life.)
  - All that decentralized peer-to-peer stuff you might have been hearing about on popular forums and what not? UUCP was already doing all that in the 70's. Where have you been?
- The network model is based on calls, and is potentially slow but robust when setup properly and doesn't require a system to be available 24/7.
  - Days or weeks can pass between calls and things will get where they need to be eventually.
  - Of course, if you are not using dial-up modems or 115200bps serial ports then you shouldn't have to wait that long.
    - But you *could* use those if you wanted to or *needed* to. Knowing UUCP can help in unusual situations.

## UUCP's upbringing

- It was developed and used in a simpler time when your primary way of electronically talking to other computers was via phone lines or weird expensive leased lines that were basically fast serial ports on either end.
- Major universities in the 70's and early 80's cooperated among themselves and others to form a wide-area UUCP network.
  - At this time if you as a home user wanted a UNIX shell or Internet access, you often could get it from your university by dialing into their systems, or sometimes your company.
  - It faded as home Internet became a thing in the early 90's and business Internet became a thing from the late 90's onward.
  - Usenet started here.

## Taylor-UUCP

-  Most common flavor of UUCP is "Taylor-UUCP" and it's in Linux Debian repos and probably other distro repos as well.
-  Other common UUCP flavor is HDB.

- Different flavors have different configuration syntax and that's about it.
- This guide can help you specifically if you're using Taylor UUCP on a modern Linux.
  - If you `apt-get install uucp` on Debian, or a Debian-based distro, that's what you'll get, plus helpful supporting programs, including `postfix` which is involved if you want to message over UUCP.

# UUCP network and service model

Parts of a working UUCP network service:





## First: Two or more nodes

First, you need two or more multiuser UNIX systems that users log into in order to do things. We'll call these nodes

- We say “multiuser UNIX system” because UUCP typically works by one system logging into another. Lucky for us, all UNIX systems are multiuser
- Any Linux install fulfills this condition.

## Second: Nodes need to have access between each other

That access can be through the following methods:

-  serial port, either directly via a null-modem cable if the systems are right next to each other physically, or a dial-up modem can be used to enable access via POTS,
-  direct unencrypted TCP connection (like telnet – bad because unencrypted, read on),
-  some weird protocol called TLI (weird, don't bother with this),
- or something that works with the `pipe` configuration option.
  - ...which is what you'll want to use unless you use serial, because...
  -  the `pipe` configuration option lets us add encryption and access control with SSH (again, keep reading)

## Third: All the action happens during a call:

Calls are what moves things through the network.

- Caller will send a set of batched-up pending files and/or tasks to the called system
- Caller will request any files or tasks the called system had batched up earlier for the caller.
- Each call results in both sides exchanging files and tasks.
- Something goes wrong and doesn't make it over? Item stays in the queue and will be retried next call until it expires.
- Some implementation details about calls and tasks:
  - Calls are processed using the `uucico` command (installed with Debian `uucp` package). That command somehow has to be connected to the call, and there's various ways to do that (keep reading).
  - Tasks are requests by the caller to run commands on the called system. `uucico` hands those off to another executable (part of typical `uucp` package) called `uuxqt`.

- This “remote execution” structure is how UUCP does email and news transfer (the executables there are `rmail` and `rnews` also part of typical `uucp` package)

## Hol up, **THE OTHER SIDE CAN EXECUTE COMMANDS ON MY SYSTEM?**

Sounds scary in 2022, right? Well, relax. Anyway, it’s not really scary because:

- By default `uuxqt` won’t run anything but `rmail` and `rnews`.
  - You have to configure it intentionally in order to run anything else.
  - You don’t need to.
- It’s running as `uucp:uucp` which shouldn’t have privilege to do much except with files it received. Those UNIX file permissions will prevent anything crazy.
- You still probably want to limit who can do that and that’s why you use SSH and setup UUCP users and passwords properly (again, keep reading).

## Fourth: **Systems periodically call each other**

To make our nodes into a network service, they have to talk to each other periodically.

- Users can log into the UNIX system, use a command to put file transfer and emails in the local system’s UUCP queue. Then the queue will be processed during scheduled calls.
- The call does not HAVE to be scheduled really, by default when `uucp` batch commands are used, it will kick off a call immediately.
  - But you probably want to define a schedule in case something goes wrong so it can retry by itself.
  - Systems that are only reachable behind certain other systems may need to schedule calls frequently so they can fully participate.
  - Otherwise when you send or receive something you might be processing a big backlog of work with it.
- The old UUCP networks would use modems over phone lines, so they would often schedule their calls after hours. This can still be done. Yes, you might have had to wait a day or longer for stuff.
- *Because each call allows both sides to exchange files and tasks, it doesn’t matter which side starts the call and which side receives the call.*

# UUCP access model for nodes

## UUCP is really the *uucico* command – on both sides

Above, we mentioned some access methods that can be used with UUCP, and we also mentioned that the *uucico* command has to somehow be connected to the call.









*I'll clarify now that *uucico* is actually involved on BOTH sides of the call.*

So what needs to happen overall is for the caller and called-system *uucico*'s to get started/be running some sort of connection so they can see each other and talk – for the duration of the call, anyway.

This is because *uucico* is the command that's actually doing the file transfer or kicking off the tasks.

*uucico* doesn't really care how that happens or where the communication comes from on either end. When you get to *uucico*'s talking, then you're doing UUCP.

So, ways to do that are:

-  Make *uucico* listen on a serial port for requests directly.
- Make *uucico* a login shell for an account. This makes *uucico* run automatically when the calling *uucico* logs in with that username and password.
  -  The login can be presented over a serial port (very classic), or over SSH. (Also TCP directly with telnet but don't do that in 2022).
  -   SSH key files have a cool feature – a key file can have a command built-in to it, such that a given key file will only allow running the command in the key file. This is better and recommended.
-  Make *uucico* talk over TCP socket.
  -   No encryption unless you add it with *stunnel* but SSH is simpler and what you should do.
  -  A classic UNIX program called *inetd* can listen for traffic on UDP or TCP ports and start programs (and give them the incoming data as stdin) when traffic is received. This is also an option.
- Even more possibilities we don't talk about in this guide.

## Just in case you're really going to use a modem

If your serial port has a modem attached, then you have to deal with dialers, because modems don't do anything useful until they dial a number. Unlike your smartphone.

Any modem you try to use in 2022 is likely Hayes compatible (most were Hayes compatible since like 1982) and a dialer config for Hayes is provided in the Debian package, so no big deal I would imagine.



## Preferred access method: SSH


The SSH method is the one talked about in this guide and the best because:

- You don't really need `uucico` running all the time as a daemon process. It would be worth it if your UUCP traffic was high enough where it mattered—and if it does, you can use something more sane for file transfer and messaging, like *anything else*.
- SSH is secure (it even says so in the name).
- SSH doesn't have certificate authorities (CAs) which makes it more convenient than SSL/`stunnel`.
- SSH's key-file system removes passwords from the login loop and enables you to control exactly who has access. Well, kinda, at least from a site-to-site perspective.

You may recall that we talked about a `pipe` configuration option to enable SSH. We'll get there, I promise.

## Serial port (modem) login

Serial port login is also discussed here because it's cool to have a network that doesn't have TCP/IP as a dependency in 2022.

- You could make `uucico` listen on the serial port directly, but having a generic login via the serial port is pretty gnarly.
  - That means you don't have to dedicate the port to `uucico`. You can log in via serial and use your Linux box in the same way people in the 70's did it.
-  You *could* do TCP/IP over a serial port using PPP, and do SSH over that.
  - I've not ever done that.

## I was thinking of trying to use the TLI protocol...

NO. Do literally anything else with your life.

# UUCP Access And File Security

You don't need to read this in order to use or setup UUCP, but you can if you are curious about security considerations when working with UUCP.

## Nodes Must “Know” Each Other Beforehand

... and...

## SSH Key Files Prevent Random People From Trying To Guess names

... both of those are true.

The name of the UUCP node can be the UNIX hostname, or defined in the UUCP `/etc/uucp/config` file.

The `/etc/uucp/sys` file must contain the name of each UUCP system that the node will call or receive a call from. This is how the nodes “know” each other.

When a call starts, the receiver's `uucico` tells the caller its name, and vice versa.

If the receiver doesn't know the caller or vice versa, `uucico` says “You are unknown to me” and drops.

- Can UUCP nodenames be spoofed? Absolutely, but because your `uucico` is only accessible via SSH and is additionally protected by a key file, you shouldn't have to worry about randos trying to guess system names.
- ❌ You weren't *really* going to directly expose it via TCP, were you?

Setting up the names in `/etc/uucp/sys` is a manual process. There is no UUCP discovery protocol, lol.

- You can't participate a UUCP network unless invited and the UUCP administrator adds your name to a node's `/etc/uucp/sys`.

## *uucico* Doesn't Run As Root

`uucico` normally runs as the user `uucp`, under the group `uucp`.

The commands `uucico` spawns for tasks also don't run as any other user (it can't because it doesn't run as root).

So, it would be difficult for someone to use UUCP to do weird things to your system unless its configured incorrectly. If you follow this guide, things will be configured correctly.

## ***uucico* Has It's Own Username And Password File At */etc/uucp/passwd***

*uucico* can be set to ask for a password itself once it is started.

That password it is looking for comes from */etc/uucp/passwd* and not the system */etc/passwd* file.

- Historical info: In the early days of UNIX, passwords were directly stored in */etc/passwd*, so it was possible at some point for any program to use it to provide logins.
  - However, modern UNIXes don't store the password there, but instead store a hash to the password in a separate file (*/etc/shadow*), unreadable by anyone but root, so that can't be done anymore.

So, once the SSH login completes and *sshd* launches *uucico* (under the user/group *uucp:uucp* of course), then if the caller doesn't provide a valid username and password **from */etc/uucp/passwd*** on top of that, things will drop.

- Usernames and passwords used when *calling* another system are defined in */etc/uucp/call*.
- If *uucico* is setup as the login shell over a serial connection, then authentication is taken care of by UNIX and *uucico* doesn't have to present a username/password prompt itself.
- If you can't tell by now, *uucico* is very flexible and will probably work over two tin cans and a string.

Just to make sure you understand: *uucico* (from Taylor-UUCP anyway) does not use the *system* *passwd* file at */etc/passwd* and couldn't do much with it if it wanted to because the passwords are hashed and shadowed in */etc/shadow*.

- There is no way to get a normal shell through *uucico* unless you allow */bin/bash* or similar as an executable command.
- There is no way to do anything as root through *uucico* unless you really go out of your way to do it, e.g. set it as your root account shell or something else really dumb.

## **UUCP Received/Sent Local File Permissions Are “free for all”, Though**

So, awesome, no one is going to break into your Linux box via UUCP, but what about transferred files?

**Transferred files go into a public area** - */var/spool/uucppublic* and by default *anyone* (777) locally can access any file there, regardless of who it is intended for.

- Files sent to specific UUCP users identified in */etc/uucp/passwd* are placed in directories that match the username.

- The `uupick` command can be used to copy them over, or you can just use `mv`.

This is where UUCP sort of throws its hands up and says “I got the files to you, it’s on you now.”

In practice this is fine unless you have unknown people on your UNIX/Linux box.

If you do: You’ll need to reconfigure things to have your files dumped somewhere inaccessible to everyone, make some shell scripts to deliver files to users, and keep users out of the `uucp` group.

If you utilize the above free-for-all mode, and a file needs to remain private between you and a specific user, a possible solution: *encrypt* it using any of the many tools available for that purpose, and communicate the decryption password or key using a method that isn’t UUCP.

# How To Use UUCP

This will tell you how to use UUCP to send/receive files.

Setting up individual nodes or multiple nodes in a UUCP network is described in later sections.

## UUCP Addresses

UUCP addresses look like the below:

```
'nodename!nodename-2!nodename-3!username'
```

The single quote marks are not technically required by UUCP, but your shell will probably need them (if you use bash) to avoid interpreting the exclamation points as some bash feature I don't use that much but probably should.

- “nodename-2” and “nodename-3” are optional, and you can actually specify as many nodenames as you want.
- The RIGHTMOST thing on the exclamation point train is a UUCP *username*, that is expected to exist on the node DIRECTLY TO THE LEFT of it.
- So in the above example, “username” must be a valid username on “nodename-3”.
- The first nodename on the LEFTMOST side must:
  - appear in your `/etc/uucp/sys`,
  - and its entry there must refer to a working port defined in `/etc/uucp/port`,
  - and have a valid username and port defined in `/etc/uucp/call` (the right way) or the username and password must appear in `/etc/uucp/sys`, with the system definition (the wrong way).
    - None of that is as bad as it sounds, it will be discussed further on.

## Simple And Usual Example

```
'someUUCPbox!someUser'
```

Above, you're asking your local UUCP to give something to the user `someUser` on `someUUCPbox` as listed in your `/etc/uucp/sys` and `/etc/uucp/port` files.

This is the usual situation.

## Via Intermediate - Forwarding

When forwarding is involved, then you have multiple nodenames:

```
'someUUCPbox!AwesomeUUCPGateway!awesomeUUCPuser'
```

What happens here is:

- Your UUCP will tell the leftmost node, `someUUCPbox`, that “I have a file for `awesomeUUCPuser` on `AwesomeUUCPGateway`.”
- `someUUCPbox` will take the request and hold it until its next call with `AwesomeUUCPGateway`.
  - Assuming you have proper access to `someUUCPbox`.
  - Of course, `AwesomeUUCPGateway` must have forwarding for `someUUCPbox` enabled. Forwarding is not enabled by default and you don’t have a way to remotely know whether or not it is enabled.
- When `someUUCPbox` has it’s next call with `AwesomeUUCPGateway`, it will relay the object with the original username, along with other queued stuff.
  - Assuming `someUUCPbox` has proper access to `AwesomeUUCPGateway`, which it should since someone setup forwarding on it.

Can there be multiple paths to a node? Yes.

Does it matter which one you specify? No, other than call schedules and bandwidth.

Can arguments between UUCP administrators affect the network map? Yes.

## Local Permissions In Order To Use UUCP Commands

Your UNIX user account must be in the `uucp` group or be root, **but don’t UUCP as root.**

To add it, someone with a root account should do `addgroup [your UNIX username] uucp`.

If you don’t do this nothing will work.

### ***uuto* - Sending A File**

Easy way is using the `uuto` command

```
uuto filename 'uucp-path'
```

That is it.

On Debian (by default) this will make a call happen almost immediately unless configured otherwise or another call happened within the last hour.

### ***uulog* - Did It Work?**

The `uulog` command will spit out recent lines from the UUCP log and show you what happened during the last call or two.

Again you have to be in the `uucp` group for that to work.

One thing you'll probably find out fairly quickly is that if a UUCP session doesn't go through, a delay is enforced before the next attempt. So you'll want to force a call – keep reading.

## Receiving Files

Do nothing. You'll get received files when:

- a system calls your UUCP, if you've setup your UUCP to be called and are known to other UUCPs, or
- when your UUCP calls a system that has files waiting for you, which it should be doing on a schedule if you are supporting the service as a whole

So, to receive files, you don't HAVE to accept calls. If you participate in a multi-mode UUCP network, then at least one node in a UUCP network must be able to receive calls in order for a service to exist, but it doesn't have to be everybody.

However...

### ***uupoll* – Call Now**

`uupoll` will make your UUCP call another system immediately, and also show you the logs for that call in realtime.

You may need to be root for this one, and may need to invoke it as `/sbin/uupoll`. Follow the command with the name of the system you want to call as listed in `/etc/uucp/sys`. Example:

```
/sbin/uupoll SomeUUCPSystem
```

If `SomeUUCPSystem` has stuff for you, you'll get it.

# Installing and Configuring Your UUCP

## Receiver And Sender Roles

As discussed above, UUCP works through systems individually calling each other. One side starts the call and the other side receives it. A call is able to be productive for both sides irrespective of who started the call.

That means there are two roles, which I'm defining here (not part of official UUCP terminology or anything like that):

- CALLER role – making your UUCP start a call to another system X
- RECEIVER role – making your UUCP able to receive calls from another system X

Note the “another system X” – because for each system X you want to enable the roles for, you will have to perform the setup.

So, for either or both roles, you'll need to do some configuring. Caller and receiver can be setup on the same UUCP or split between different UUCPs.

## Preparing for either role

### 1. Install UUCP If You Need To (Assuming Debian)

On Debian this will be, as root:

```
apt-get install uucp
```

It will want to pull in postfix, and that's OK. postfix is used to enable the Linux mail commands to send messages via UUCP. You'll get a dpkg-reconfigure screen asking how you want to setup postfix. **Select “No configuration” for now.**

### 2. Read This And Understand How SSH Keys Work

Review the below information and make sure you understand it before continuing.

Reference: <https://askubuntu.com/questions/46424/how-do-i-add-ssh-keys-to-authorized-keys-file#>

Excerpt:

You should never save the file with its contents starting with -----BEGIN RSA PRIVATE KEY----- on the server, that is your private key. Instead, you must put the public key into the ~/.ssh/authorized\_keys file.

This public key has the .pub extension when generated using ssh-keygen and its contents begin with ssh-rsa AAAAB3. (The binary format is described in the answers to this question).

The permissions of ~/.ssh on the server should be 700. The file ~/.ssh/authorized\_keys (on the server) is supposed to have a mode of 600. The permissions of the (private) key on the client-side should be 600.



### 3. Decide A Few Things

Figure out what you want your nodename to be, or know the nodename of the system you're going to call. Below we'll refer to that using `YOUR_NODE_NAME`.

- Callers – make sure you know the nodename as defined in the config files of the system your UUCP will be calling.
- Receivers –
  - If your UUCP will be accessible via the Internet, and you have some sort of DNS pointing to your IP, you can use that.
  - Otherwise you can use the hostname of your system, or make up something convenient.
  - UUCPs ask and expect others names as defined in config files so it matters, but does not have to match any DNS or your UNIX hostname.

## SSH RECEIVER role setup step-by-step

### 1. Create A Local User Just For Incoming UUCP Requests

On my system I called that user `uucp-external`.

Also: I like to put the home directories of “system” users in `/etc/local/servicehome` but that is just my preference, and you can set that to be wherever (default `/home` is fine)

```
# adduser --ingroup uucp --home /etc/local/servicehome/uucp-external --uid 400  
--disabled-password uucp-external
```

### 2. Create SSH Key Files

#### *A Note On Key Hygiene*

If you read the excerpt at the beginning of this guide, you should recall that it said to **never store the private key on your server**. You really shouldn't—you will be giving it out to receivers but it definitely should not stay in this spot once you are in “production.”

- If you want to be hardcore, you can have a USB drive or other external storage connected, `cd` to a directory that lives on that storage, and generate the files there.
- If you want to be better but less than hardcore, generate these keys and move the private key away from a location accessible by `uucp-external`.

Login as the user temporarily (easiest way)

```
# su uucp-external
```

Then:

```
$ ssh-keygen -C "uucp-external@YOUR_NODE_NAME" -f ~uucp-external/YOUR_NODE_NAME.key
```

Then, press Enter twice (no passphrase). It should make two files, `YOUR_NODE_NAME.key` and `YOUR_NODE_NAME.key.pub`.

### 3. Putting The Key In The Right Spot For *sshd*

Run all these commands. Doing something wrong here will make SSH act like the key file doesn't exist and want a password.

```
$ cd ~
$ mkdir .ssh
$ chmod 700 .ssh
$ cd .ssh
$ touch authorized_keys
$ chmod 600 authorized_keys
```

### 4. Setting Up *.ssh/authorized\_keys*

Open the `YOUR_NODE_NAME.key.pub` file (not the plain `YOUR_NODE_NAME.key`) file in a text editor and copy all of it.

Then open the `authorized_keys` file and paste it in there.

Next, add the following around that key you pasted to make your `.ssh/authorized_keys` file look like this, then save it.

```
no-port-forwarding,no-X11-forwarding,no-agent-forwarding,command="/usr/sbin/uucico -l"
ssh-rsa AAAB3BIG_STRING_OF_RANDOM_KEY_TEXT uucp-external@YOUR_EXTERNAL_DOMAIN
```

If you want to delete the public key file you can at this point.

### 5. Disable Password Authentication For Your UUCP Dedicated User

You want to disable password authentication for the account you have dedicated to UUCP.

Do that by adding the following to the end of your `/etc/ssh/sshd_config`:

```
Match User uucp-external
    PasswordAuthentication No
```

## 6. Define Some UUCP Users And Tell UUCP Your Chosen `nodename`

Add a username and password to `/etc/uucp/passwd` - it does not have to match any existing Linux usernames.

Callers will need to use these usernames and passwords, otherwise your `uucico` will give them the cold shoulder.

Reminders:

- `uucico` is the executable that uses this file.
- Keep in mind that `uucico` runs as the system user `uucp` and not `root`, so anything `uucico` does can be blocked with Linux permissions fairly easily, and by default `uucico` does not have free reign of your system.
- Don't get scared just because this file has `passwd` in the name.

Then edit `/etc/uucp/config` and change the `nodename` to `YOUR_NODE_NAME`.

## 7. Add Systems That Will Call You To `/etc/uucp/sys`

See step 1a for SSH CALLER role setup step-by-step below.

# SSH CALLER role setup step-by-step

## 1. Get That Private Key File From The Receiver.

Do that.

### 1a. Ask The Receiver To Add Your nodename To His/Her `/etc/uucp/sys` File

If your nodename is `awesome-node` (defined in `/etc/uucp/config`), then the receiver needs to append the following to his/her `/etc/uucp/sys` file.

```
system awesome-node
protocol i
```

That takes care of the receiver “knowing” the caller, which is required.

The receiver doesn’t need further information unless it plans to start calls with you. If it does, it should be following these steps vice versa, with your information.

## 2. Put The Receiver’s Private Key File In A Place *uucico* Can See And Read It

Recommended location is `/etc/uucp`.

If the sender followed the above steps, you should have a file like this:

```
/etc/uucp/RECEIVER_NODE_NAME.key
```

Now do these two things...

```
$ chmod 600 "/etc/uucp/RECEIVER_NODE_NAME.key"
$ chown uucp:uucp "/etc/uucp/RECEIVER_NODE_NAME.key"
```

...to that file.

SSH will check the permissions, and the `uucp` user needs to be able to read it.

- This does mean that anyone in the `uucp` group could get these private key files.
  - That’s why the receiver connects the private key to a dedicated public user with limited capability.
  - That’s also why you only put people in the `uucp` group that you trust.

## 3. Edit Some UUCP Configuration Files

**`/etc/uucp/sys`**

Append the below to `/etc/uucp/sys`. You will have to be root to make changes to the file (the `uucp` user can read it but not alter it).

```
call-login *
call-password *
time any
chat "" \d\d\r\c ogin: \d\L word: \P
chat-timeout 30
protocol i
port SSH-YOUR_NODE_NAME
```

### ***/etc/uucp/call***

Add a UUCP username and password from the receiver system to this file. You'll also need to specify the node name of the receiver system.

### ***/etc/uucp/Port***

You'll need to add a port to the UUCP config. Here's where the receiver's private key is referenced.

You will also need to specify the **receiver's actual DNS domain name or IP** where it says **RECEIVER\_DOMAIN\_NAME\_OR\_IP**.

So, append the following to `/etc/uucp/Port`:

```
port SSH-RECEIVER_NODE_NAME
type pipe
command /usr/bin/ssh -a -x -q -i /etc/uucp/RECEIVER_NODE_NAME.key -l uucp-external
RECEIVER_DOMAIN_NAME_OR_IP
reliable true
protocol etyig
```

## **4. Log Into The Receiver Once Manually With The SSH Command In The Port File**

Just copy and paste the `ssh` command and run it once.

```
$/usr/bin/ssh -a -x -q -i /etc/uucp/RECEIVER_NODE_NAME.key -l uucp-external
RECEIVER_DOMAIN_NAME_OR_IP
```

You'll see the usual prompt about a previously unknown system, say Yes. You should then connect and get a `login:` prompt.

Aren't you already logged in through the private key? Yes, but that was just the private Uber/Lyft to the front door. Now you're talking to the "receptionist" who wants to know who you are – the receiver's `uucico` is talking to you now, and it wants a UUCP username and password.

That comes from that `/etc/uucp/call` file.

Since we are just testing, you can disconnect (or enter the UUCP username and password if you really want to keep going).

At this point, great job! The link is validated working, and `uucp` group members should be able to call the receiver at your convenience.

## **5. The Receiver Doesn't Need Something In `/etc/uucp/Port` For My System, As A Caller?**

No. *Ports are for callers.* Receiver end of access is taken care of through the SSH mechanism using this scheme and defining a port on the receiver end is not needed.

# Additional UUCP stuff

## UUCP Node Forwarding

It is possible for a caller to send a file to a user that is not on the receiver, but a system that the receiver knows about and will call or receive calls from.

Enabling this is simple, and done in `/etc/uucp/sys`.

There are two options – `forward-from` and `forward-to`, and each option will take a space-separated list of systems that it will forward things from, or forward things to.

- The systems listed in `forward-from` and `forward-to` must also appear in the `/etc/uucp/sys` file as a known system.
- Forwarding has nothing to do with the call out schedule on that system, if forwarding is enabled, it'll happen on the next call with the specified systems no matter who called who.

Example:

```
nodename hubNode  
  
forward-from remoteNode1 remoteNode2  
  
forward-to remoteNode1 remoteNode2
```

With the above in `/etc/uucp/sys`, `remoteNode1` can send something to `remoteNode2` *through* `hubNode`, `remoteNode1` would tell `HubNode` in a call that it wants to send something to `remoteNode2`. On the next call with `remoteNode2`, `HubNode` will give `remoteNode2` the file `remoteNode1` wanted to send.

Calls asking for things not on the same UUCP system will be denied if forwarding options aren't in `/etc/uucp/sys`.

Forwarding files for other UUCP nodes is something you shouldn't do unless you like and trust the other operators.

# UUCP mail setup

## Traditional UNIX Mail Workflow (Where Email Was Born)

Brace yourself, this will get a bit involved.

Before the microcomputer revolution of the mid-70's and the home computer/desktop PC revolution of the early 80's, all computers were big things designed to be accessed via text terminals by multiple users at once.

It made obvious sense for most operating systems running on these types of computers to have infrastructure for users to leave messages for one another. Such systems often had multiple people logged in simultaneously, and they might even actually be collaborating on stuff.

UNIX is no exception.

The history of UNIX mail is complex, but all you need to know is that it's more or less the standard for a UNIX install (including Linux) to have a `mail` command that works like this

```
echo "Body of e-mail" | mail -s "Subject" someone@somewhere
```

What does that command actually do with the email you're giving it via a UNIX pipe and command line options? Well, before SMTP it would just copy messages into other local users mail folders, and also users could use it to read messages off of their own mailbox and even send email.

Excerpt from the [Wikipedia article on mail \(Unix\)](#):

This version of mail was capable to send (append) messages to the mailboxes of other users on the Unix system, and it helped managing (reading) the mailbox of the current user.

And that excerpt talks about First Edition UNIX from 1971. So... email and UNIX have gone hand in hand since UNIX was created.

More history:

- Fast forward to bit later to the late 70's...
  - ...that's about the time UUCP, and the `rmail` command came into being.
  - ...also somewhere around that time as well there was such a thing as `delivermail` which could use... get this, FTP to transfer email (FTP dating from 1971).
    - Yes, FTP is damn old.
- Then... SMTP. When SMTP became a thing in the early 80's, `sendmail` joined the party, which would route mail outside of the system using the SMTP protocol if needed, and if not needed, do the same thing as classic `mail`.
  - The role `sendmail` plays in the UNIX mail workflow is called the *Mail Transfer Agent* or MTA.



- `sendmail` supported UUCP along with SMTP. (Not sure about FTP, lol.)
- **Warning: You never ever want to mess around with `sendmail` in your life, not this one or your next one. You need to be on multiple types of drugs simultaneously to successfully create a working `sendmail` configuration file. DO NOT GO THERE.**

Postfix is a modern MTA that is `sendmail` compatible (as are most UNIX/Linux MTAs), supports UUCP, and has sane configuration syntax (*potentially* complex, but you don't need to be on drugs to work with it).

And now you know why `postfix` was installed along with `uucp` when you did the `apt-get install uucp`.

## It's Sounding Like We Gotta Setup *postfix*

Yep, it's pretty much necessary as far as I know to integrate with the traditional UNIX mail workflow these days, even if you don't want to use SMTP.

The benefits are once it's working, you can use command line Linux email tools to read, compose, and send messages, the same ones you'd use with SMTP.

When you install `uucp` on Debian, `postfix` will come with it.

Postfix provides all the tools to enable a standard UNIX mail workflow on your system, and it also more or less assumes you're living in at least the 90's and want to use SMTP, but it does still support UUCP in there.

You can even turn off network protocols in `postfix` and go full UUCP which is what we're going to do below.

## All right, let's Do This: Configuring Postfix For UUCP

If you've been following this guide, you installed `postfix` and selected 'No Configuration' on the 'dpkg-reconfigure' menu.

- If you didn't, `apt-get remove postfix`, `apt-get purge postfix`, then `apt-get install postfix` again. Then be sure to select 'No Configuration' on the 'dpkg-reconfigure' menu.

You now have a totally blank Postfix configuration, which won't even work because some required things are blank.

### 1. *master.cf*

Find this line and comment it out so it looks like this:

```
#smtp      inet  n       -       y       -       -       smtpd
```

Turning off SMTP is that simple

- Now, if you have an existing SMTP based setup with Postfix that's working that you want to keep, then don't do this. If you barely know what SMTP stands for and you haven't been working with Linux too long, ignore this.

There's another line two lines down that begins with `#smtpd` – that's fine how it is.

Then look for the line that has `uucp` in it and UNcomment it - that means **remove the #**.

```
uucp      unix  -      n      n      -      -      pipe
```

## 2. *main.cf*

Change the `myorigin` = line to the name of your UUCP node.

```
myorigin = CoolUUCPEmailNode
```

Change the `mydestination` = line to the same thing.

```
mydestination = CoolUUCPEmailNode
```

Towards the bottom now, these will be blank, fill them in with these exact values

```
sendmail_path = /usr/sbin/sendmail
newaliases_path = /usr/bin/newaliases
mailq_path = /usr/bin/mailq
setgid_group = postdrop
```

`inet_protocols` should be set to blank because we want to make sure outgoing mail only goes through UUCP and not SMTP, for the purposes of this guide anyway.

```
inet_protocols =
```

Last, add these lines:

```
default_transport = uucp
transport_maps = hash:/etc/postfix/transport
smtp_dns_support_level = disabled
lmtp_dns_support_level = disabled
disable_dns_lookups = yes
```

Yes, we don't want Postfix doing DNS lookups if we are using UUCP only, because UUCP is

responsible for starting the SSH connection and will basically run **/bin/ssh** which does DNS on its own.

- If you don't, Postfix will try to look up your UUCP nodenames in DNS and will not send mail because it doesn't resolve.

Above, we're trying to make Postfix simply kick off UUCP when it's asked to send a mail, and forget anything about SMTP.

### 3. Make An **/etc/postfix/transport** File

The transport rule will let us define email routing rules--what is done depending on the email address or what's on the right side of the **@**.

This is how we control postfix and avoid it treating our UUCPs like a public email server which will accept anything.

The file should start with a line like this

<b>YOUR_UUCP_NODENAME</b>	<b>local</b>
---------------------------	--------------

Honestly, you probably want a few of these lines.

<b>YOUR_UUCP_NODENAME</b>	<b>local</b>
<b>YOUR_HOSTNAME</b>	<b>local</b>
<b>localhost</b>	<b>local</b>
<b>127.0.0.1</b>	<b>local</b>

The above line will enable local users to email each other (and covers common things email programs do) and Postfix won't try to make that go through anything external.

Concrete example: Let's say you have 3 local users, john, mary, and mike. The above will make this work without generating a UUCP call.

```
john$ echo 'test' | mail -s 'test' mary@YOUR_NODE_NAME
```

Awesome, that takes care of local users, but ... we're setting up UUCP here. Don't we do something so we can get mail to cross over to our UUCP friend?

Yes, and it looks like this – add another line:

<b>OTHER_UUCP_NODE_NAME_1</b>	<b>uucp:OTHER_UUCP_NODENAME_2</b>
-------------------------------	-----------------------------------

What this does is make this...

<b>john\$ echo 'test'   mail -s 'test' someone@OTHER_NODENAME_1</b>
---

send over UUCP to OTHER\_UUCP\_NODENAME\_2 – and OTHER\_UUCP\_NODENAME\_2 should be a remote node defined in /etc/uucp/sys and able to make an outgoing call.

You probably want OTHER\_UUCP\_NODENAME\_1 and OTHER\_UUCP\_NODENAME\_2 to match – but OTHER\_UUCP\_NODENAME\_1 can be anything you want really.

- OTHER\_UUCP\_NODENAME1 is the domain users on that specific system will use to send emails, and
- that precious transport file maps it to OTHER\_UUCP\_NODENAME\_2, which tells Postfix what to contact.

To keep things as simple and as straightforward as possible, you should have them match.

Also, keep in mind:

- Yes, the mail command (and other standard mail tools that also work with SMTP email) wants an @ between the username and UUCP node name, not an exclamation point.

#### ***4. Postfix Actually Wants Its Transport File In A Database Format.***

...so one more step.

Postfix comes with a postmap command that converts the transport file we just made to a database.

```
# /sbin/postmap /etc/postfix/transport
```

Then, do this because it might be needed for weird reasons

```
# /sbin/newaliases
```

Then, restart Postfix to make it recognize all those changes.

```
# /sbin/postfix reload (or systemctl restart postfix)
```

Need to do that anytime the transport file is updated.

#### ***5. What About The Other Side?***

##### **Other Side Wants To Send And Receive With You**

If you want the other side to be able to **send and receive email over UUCP with you**, the other side has to do these steps too, putting your UUCP nodename in /etc/postfix/transport.

##### **Other Side Just Wants To Receive**

If the other side simply wants to **receive email** over UUCP, then:

The other side needs a Mail Delivery Agent (MDA) installed, but not an MTA.

An MDA will take email coming from commands or programs on the local system, and drop it in local users mailboxes.

- That goes way back to when that's all there was - local mail.
- When an MTA (thing receiving email from outside) gets some email from outside that's destined for a local user, it talks to the MDA in the same manner.

Great news! `postfix` is not only an MTA, but an MDA as well. So by Debian automatically installing it, you're covered there.

And to setup `postfix` to handle MDA duties you have to do ... not a lot.

On the "receive only" end, you still want to comment out this line in `master.cf`:

```
#smtp      inet  n       -       y       -       -       smtpd
```

and do a `/sbin/postfix reload`. That will turn off the SMTP server so no concern about unnecessary Internet programs running.

Then these still need to be set:

```
sendmail_path = /usr/sbin/sendmail
newaliases_path = /usr/bin/newaliases
mailq_path = /usr/bin/mailq
setgid_group = postdrop
```

And that's it.

UUCP email reception works by the sender asking the receiver to remotely execute a command called `rmail`. `rmail` will take what's coming in from the UUCP link (piped there by `uucico`) and hand it to the local MDA - kinda similar to that basic `mail` command.

So ... again, you don't even need to tell postfix about any UUCP nodes unless you want to send email to them.

## 6. Testing

If you followed the above to get Postfix to send email over UUCP, here's how to test.

1. Send a mail - `echo "test" | mail -s "test" any-local-user@remote-uucp-nodename` will do.
2. Do a `sudo /sbin/mailq` and check Postfix's queue. If everything is OK it should say Mail queue is empty.
3. What Postfix should have done is get something in the UUCP queue. So let's check that. Do a `/sbin/uulog`.

4. If everything is good, you'll see some stuff about transmission, or possibly something saying UUCP has queued it up. If it's queued, make it go with `/sbin/uupoll remote-uucp-nodename`. You'll get a live play-by-play of UUCP sending the message (and anything else queued up) over.
5. Now go to the remote system as `any-local-user` and check your mailbox. You should receive the message. If something like `alpine` or `mutt` isn't installed the plain mail command can be used (without any parameters) to take a quick look.
6. Did `any-local-user` not actually exist on the other system as UNIX/Linux user? Then you're going to get a classic MAILER-DAEMON bounce message. That's your fault.

## 7. Setup Errors You May Run Into And Fix Suggestions

### **mailq shows an error about address resolution failing**

Postfix assumes you are a sane person and trying to send your email using something using that newfangled Internet technology that uses IP and DNS, like SMTP.

Even though you probably have UUCP going over SSH which *does* use the Internet, UUCP is an old angry curmudgeon and doesn't want UUCP's help.

- We don't want *postfix* to talk to the Internet directly really - we want Postfix to just tell UUCP to reach out to system X (and that's defined in that `transport` file)
  - Then UUCP will handle it.
    - When you tell UUCP to contact system X, it looks up things in the `sys` and `Port` files, and then runs the SSH command - and SSH will work over the Internet like normal. But if that `Port` file contained info for a modem or serial port setup, it would still work and Postfix can really wash its hands once it talks to UUCP.

So what you can do is this:

- Disable DNS in Postfix with these configuration items in `/etc/postfix/main.cf`:  
**`smtp_dns_support_level = disabled, lmtp_dns_support_level = disabled, disable_dns_lookups = yes`**
- Or put fake entries for your UUCP nodes in your `/etc/hosts` file. Just the nodename, `no.local`, `.localhost` or anything like that. Should fool Postfix enough to stop complaining.

### **uulog Shows Some Complaining About Unknown User**

Did you ...

- try to email a username from your `/etc/uucp/passwd` or `/etc/uucp/call` file?
- or try to email a username that isn't a local UNIX/Linux account on the other side?

Well, that won't work unless the other side has a local UNIX/Linux account by that name.

For testing, try `root` or `postmaster`.

### **uu1og Says Something About Error 75**

One thing: your email message might be too large. Adjust the maximum message size in Postfix.

### **External Email Clients**

So if you've been following everything, you'll be aware that Postfix will drop received email in a UNIX/Linux mailbox.

That means you have to be logged in and run an app on that local Linux box to read email, 1970's style.

Not too bad, but you may be wondering how to connect an email client to it?

Email clients talk IMAP. So you need to run an IMAP server. The IMAP server I use is Dovecot. I won't cover Dovecot here, sorry.

## Serial port-based nodes

It seems silly in 2022, but you can do the following if you wanted:

- Connect two Linux PCs via a null-modem serial port cable.
  - Typical name Linux gives the first serial port is `/dev/ttyS0`.
- Tell `systemd` to keep a `getty` running on a serial port device, such as `/dev/ttyS0`.
- The result: if you use a terminal program, such as `minicom`, on the other PC, and hit Enter a few times, you will get a direct login prompt.

It is honestly a great way to ensure access to a system outside of SSH in case your network or some other network device fails, such as a switch.

You can also leverage it for UUCP.

## No Keys With Serial

We don't do anything with keys when utilizing serial setup. It's not needed. If the concern of someone breaking into your house and tapping your serial port (entirely possible and easy), then don't use this method. ☺



## Serial RECEIVER role setup step-by-step

### 1. Create a user just for incoming serial UUCP requests (separate from SSH-based user!)

On my system I called that user `uucp-external2`.

If you are receiving calls from both SSH and serial, do not use the same user.

Also: I like to put the home directories of “system” users in `/etc/local/servicehome` but that is just my preference, and you can set that to be wherever (default `/home` is fine)

```
# adduser --ingroup uucp --home /etc/local/servicehome/uucp-external2 --uid 401
--disabled-password uucp-external2
```

### 2. Set login shell to *uucico*.

Edit `/etc/passwd` (yes, “the” `/etc/passwd`) and look for a line with `uucp-external2` in it. Make it look like this:

```
uucp-external2:x:401:10::,/etc/local/servicehome/uucp-external2:/usr/sbin/uucico
```

### 3. Give incoming serial UUCP user a really strong password

32 random characters of letters (mixed case) and numbers at a minimum. Remember this password.

### 4. Tell *systemd* to maintain a *getty* on `/dev/ttyS0`

This—<https://www.rogerirwin.co.nz/open-source/enabling-a-serial-port-console/>--tells you.

Excerpt:

#### Linux devices with systemd

```
sudo nano /lib/systemd/system/serial-getty@.service
```

First we need to edit the `serial-getty` service to set the correct baud rate.

```
# Edit this line
ExecStart=--/sbin/agetty --keep-baud 115200,38400,9600 %I $TERM
# To This
ExecStart=--/sbin/agetty 115200 %I $TERM
```

Set this to the baud rate that you wish to use. This should be the same at the computer is using at the other end.

```
systemctl daemon-reload
# For a USB serial adaptor
systemctl enable serial-getty@ttyUSB0.service
# For a built in serial port /dev/ttyS0
```

```
systemctl enable serial-getty@ttyS0.service
```

You can now log in using the serial console on `/dev/ttyUSB0` or `/dev/ttyS0`

## **5. Add the system that will call you through serial port to `/etc/uucp/sys`**

There should be only 1 possible system that can call you through that port.

Same deal as for SSH-based callers. See 1a for CALLER setup above.

# Serial CALLER role setup step-by-step

## 1. Receiver's nodename must appear the `/etc/uucp/sys` file.

If your nodename is `awesome-node` (defined in `/etc/uucp/config`), then the following needs to be appended to the receiver's `/etc/uucp/sys` file.

```
system awesome-node
protocol i
```

That takes care of the receiver “knowing” the caller, which is required.

The receiver doesn't need further information unless it plans to start calls with you, which it can't because the receiver is presenting a login prompt over serial to the caller, but the caller can't do that over the same serial port at the same time.

If you really want bidirectional communication over serial, you need 2 serial ports. It's strange in 2022 but they do make PCIe serial port adapters.

## 2. Edit some UUCP configuration files

### `/etc/uucp/sys`

Append the below to `/etc/uucp/sys`. You will have to be root to make changes to the file (the `uucp` user can read it but not alter it).

```
call-login *
call-password *
time any
chat "" \d\d\r\c ogin: \d\L word: \P
chat-timeout 30
protocol i
port serial
```

### `/etc/uucp/call`

Add a UUCP username and password from the receiver system to this file. You'll also need to specify the node name of the receiver system.

### `/etc/uucp/Port`

For serial setup, you'll need to add a port definition to the UUCP config. There is one defined in the file already but it is for a modem. We'll add another one.

Append the following to `/etc/uucp/Port`:

```
port serial
type direct
device /dev/ttyS0
hardflow false
speed 115200
protocol i
```

## Configuration References

[https://www.math.utah.edu/docs/info/uucp\\_4.html#SEC36](https://www.math.utah.edu/docs/info/uucp_4.html#SEC36)

[https://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/html\\_single/UUCP-HOWTO.html](https://www.ibiblio.org/pub/Linux/docs/HOWTO/other-formats/html_single/UUCP-HOWTO.html)

[http://osr507doc.xinuos.com/en/NetAdminG/tcpT.uucp\\_over\\_tli.html](http://osr507doc.xinuos.com/en/NetAdminG/tcpT.uucp_over_tli.html)

[https://www.airs.com/ian/uucp-doc/uucp\\_6.html#SEC77](https://www.airs.com/ian/uucp-doc/uucp_6.html#SEC77)