SSH keys: log in without typing your password

SSH keys let you log in to the HPCC cluster, and push to GitHub, without typing a password (and, on the cluster, without the Duo prompt) every time. You set them up **once per computer**.

This page has a quick start if you just want the commands, then explains what the files are, where they go, and what to do when it doesn't work.

Quick start

Replace `YOURHPCCUSERNAME` with your cluster login name and use your own email.

Mac or Linux (Terminal)

```
# 1. make a key pair on YOUR LAPTOP (press Enter to accept the file name,
#    then pick a passphrase you will remember)
ssh-keygen -t ed25519 -C "yourname@ucr.edu"

# 2. copy the PUBLIC key to the cluster (asks for your password + Duo one last time)
ssh-copy-id -i ~/.ssh/id_ed25519.pub YOURHPCCUSERNAME@cluster.hpcc.ucr.edu

# 3. test it: should ask for your key passphrase (or nothing), not your password
ssh YOURHPCCUSERNAME@cluster.hpcc.ucr.edu
```

Windows (PowerShell, Windows 10 or 11)

Windows includes the same `ssh` tools. Open **PowerShell** (not the old Command Prompt):

```
# 1. make a key pair
ssh-keygen -t ed25519 -C "yourname@ucr.edu"

# 2. copy the PUBLIC key to the cluster (Windows has no ssh-copy-id, so we pipe it)
type $env:USERPROFILE\.ssh\id_ed25519.pub | ssh YOURHPCCUSERNAME@cluster.hpcc.ucr.edu "mkdir -p ~/.ssh &&

# 3. test it
ssh YOURHPCCUSERNAME@cluster.hpcc.ucr.edu
```

If you use **MobaXterm**, open a local terminal in MobaXterm and follow the Mac/Linux steps; MobaXterm has `ssh-copy-id`.

Then add the same public key to GitHub

```
cat ~/.ssh/id_ed25519.pub       # copy the ONE line this prints
```

On github.com go to your picture (top right) -> **Settings** -> **SSH and GPG keys** -> **New SSH key**, paste the line, and save. Test it:

```
ssh -T git@github.com
# Hi YOURGITHUBID! You've successfully authenticated, but GitHub does not provide shell access.
```

How SSH keys work

`ssh-keygen` makes **two files that belong together**, a *key pair*:

- the **private key** (`id_ed25519`) - this is like the key to your house. It stays on the computer where you made it. **Never copy it anywhere, email it, or put it on GitHub.**
- the **public key** (`id_ed25519.pub`, ending in `.pub`) - this is like a lock that only your private key opens. It is safe to share. You install copies of it on every server you want to log in to.

When you connect, the server checks whether your laptop holds the private key matching one of the public keys it has on file. The private key never leaves your laptop.

The **passphrase** you pick when making the key encrypts the private key file, so a stolen laptop doesn't give someone access to the cluster. It is *not* your UCR or cluster password. With the `ssh-agent` (below) you type it only once per session.

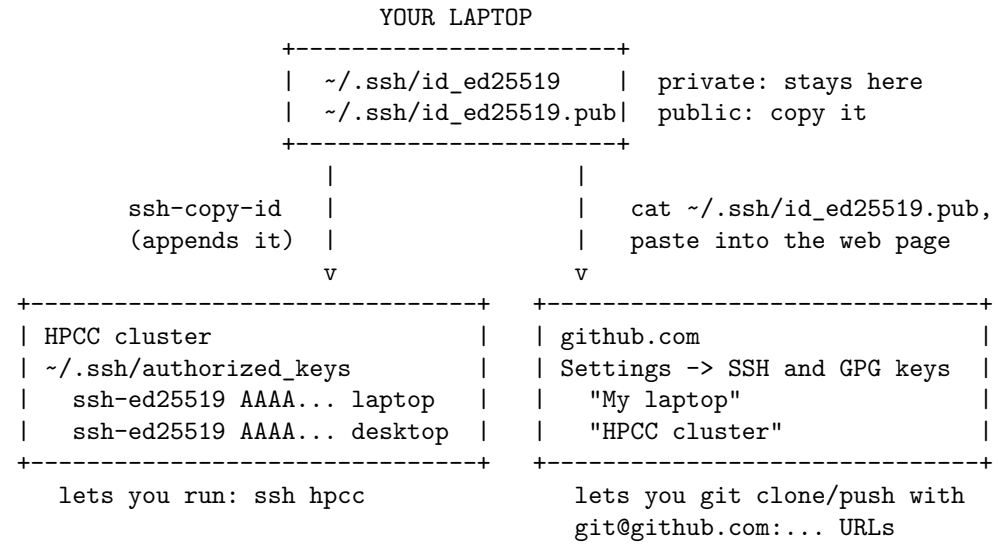
Where the files are

Everything lives in a hidden folder called `.ssh` in your home directory (on Windows, `C:\Users\YOU\.ssh`). Use `ls -la ~/.ssh` to see it.

```
YOUR LAPTOP                                HPCC CLUSTER (cluster.hpcc.ucr.edu)
~/.ssh/                                    ~/.ssh/
|-- id_ed25519        private key           |-- authorized_keys   public keys allowed
|                     (never copy it)       |                     to log in to your
|-- id_ed25519.pub    public key            |                     account, one per line
|                     (copy this one)       |
|-- config            your shortcuts,       |-- id_ed25519        OPTIONAL: a separate
|                     e.g. "ssh hpcc"       |-- id_ed25519.pub    key pair made on the
|                     (optional)            |                     cluster for GitHub
`-- known_hosts       servers you have      `-- known_hosts
                      connected to before
```

What gets copied where

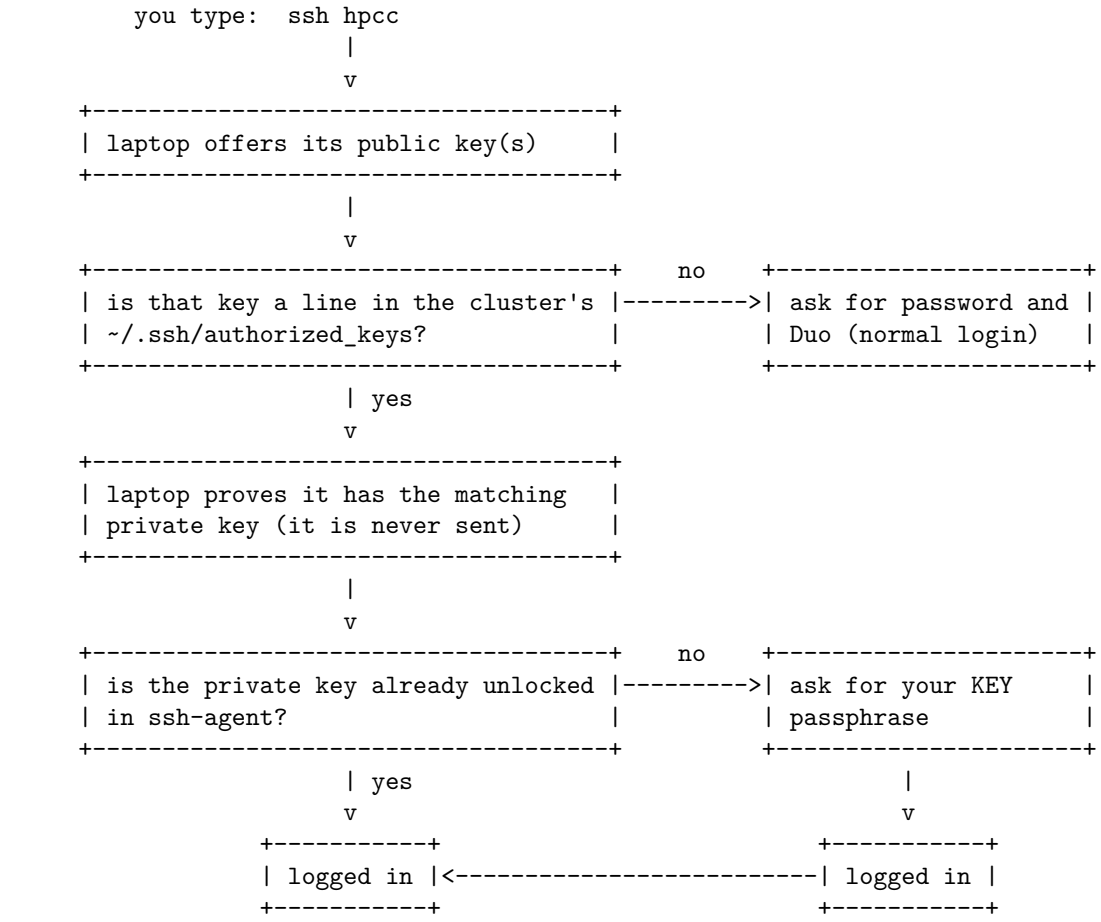
Only the **public** key (`.pub`) is ever copied. It gets **added as a line** to the cluster's `authorized_keys` file, and pasted into your GitHub settings:



A few things follow from this:

- **One key pair per computer.** Your laptop and your lab desktop each make their own pair; each public key is one line in `authorized_keys` and one entry on GitHub. If you lose a laptop, delete just that line/entry.
- **Using GitHub from the cluster** (e.g. `git push` from your homework folder on HPCC) needs a key that is *on the cluster*. Log in to the cluster, run `ssh-keygen -t ed25519` there, and add *that* `.pub` file to GitHub as a second key. Don't copy your laptop's private key to the cluster.
- The cluster's home directory is shared by all the cluster nodes, so one `authorized_keys` file works everywhere on HPCC.

What happens when you log in



If you get asked for your *password* (not your passphrase), the cluster didn't accept your key - see Troubleshooting.

Step by step

1. Make the key pair (on your laptop)

```
ssh-keygen -t ed25519 -C "yourname@ucr.edu"
```

```
Generating public/private ed25519 key pair.
Enter file in which to save the key (/Users/you/.ssh/id_ed25519):     <- press Enter
Enter passphrase (empty for no passphrase):                           <- type a passphrase
Enter same passphrase again:
Your identification has been saved in /Users/you/.ssh/id_ed25519
Your public key has been saved in /Users/you/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:u0c6sXeS6Lf94ExzEu3+pFnVYKQpuev1yoQCn0mR5W0 yourname@ucr.edu
```

- `-t ed25519` picks the key type. Ed25519 keys are short, fast and secure, and are what GitHub recommends. If some old system rejects them, make an RSA key instead with `ssh-keygen -t rsa -b 4096` (the files are then `id_rsa` and `id_rsa.pub`).
- `-C` adds a comment (label) to the end of the public key so you can tell your keys apart later.
- If it says the file **already exists**, you already have a key: answer `n` and use the one you have (check `ls ~/.ssh`). Overwriting it will break anywhere that key was set up.

The public key is a single line of text:

```
cat ~/.ssh/id_ed25519.pub
```

```
ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIMng... yourname@ucr.edu
```

2. Put the public key on the cluster

The easy way (Mac, Linux, MobaXterm):

```
ssh-copy-id -i ~/.ssh/id_ed25519.pub YOURHPCCUSERNAME@cluster.hpcc.ucr.edu
```

This logs in with your password and Duo one last time, creates `~/.ssh` on the cluster if needed, **adds** your key to the end of `authorized_keys`, and sets the permissions.

The manual way, which is what `ssh-copy-id` does for you:

```
# on your laptop: copy the public key file to the cluster under a temporary name
scp ~/.ssh/id_ed25519.pub YOURHPCCUSERNAME@cluster.hpcc.ucr.edu:mylaptop.pub

# log in (password + Duo) and append it to authorized_keys
ssh YOURHPCCUSERNAME@cluster.hpcc.ucr.edu
mkdir -p ~/.ssh
chmod 700 ~/.ssh
cat ~/mylaptop.pub >> ~/.ssh/authorized_keys      # >> ADDS; a single > would erase other keys
chmod 600 ~/.ssh/authorized_keys
rm ~/mylaptop.pub
exit
```

If you can't log in with a password at all, the HPCC staff can install your **public** key for you: email the `.pub` file to support@hpcc.ucr.edu (see the HPCC login instructions).

3. Test it

```
ssh YOURHPCCUSERNAME@cluster.hpcc.ucr.edu
```

You should be asked for your **key passphrase** (or nothing, if the agent has it), and no Duo prompt.

4. Type the passphrase once: ssh-agent

The `ssh-agent` program remembers your unlocked key while you are logged in to your laptop:

```
ssh-add ~/.ssh/id_ed25519        # asks for the passphrase once
ssh-add -l                       # list keys the agent is holding
```

On a Mac, add `--apple-use-keychain` (`ssh-add --apple-use-keychain ~/.ssh/id_ed25519`) to store the passphrase in the macOS keychain, so it survives restarts. The `config` file below does this automatically. On Windows, the agent is a service that has to be turned on once in an administrator PowerShell with `Get-Service ssh-agent | Set-Service -StartupType Automatic` and `Start-Service ssh-agent`.

5. Shortcuts: `~/.ssh/config`

Instead of typing `ssh YOURHPCCUSERNAME@cluster.hpcc.ucr.edu` every time, create (on your laptop) a text file `~/.ssh/config`:

```
# settings for every host
Host *
    AddKeysToAgent yes
    IgnoreUnknown UseKeychain
    UseKeychain yes
    ServerAliveInterval 30

# the UCR HPCC cluster: "ssh hpcc"
Host hpcc
    HostName cluster.hpcc.ucr.edu
```

```
    User YOURHPCCUSERNAME
    IdentityFile ~/.ssh/id_ed25519
    ForwardX11 yes

# GitHub
Host github.com
    User git
    IdentityFile ~/.ssh/id_ed25519
```

Now these all work, and `scp`/`rsync` understand the shortcut too:

```
ssh hpcc
scp results.tsv hpcc:bigdata/gen220/
rsync -av hpcc:bigdata/gen220/project/ project/
```

- `AddKeysToAgent yes` adds your key to the agent the first time you use it.
- `UseKeychain yes` saves the passphrase in the Mac keychain; `IgnoreUnknown UseKeychain` stops Linux and Windows from complaining about that Mac-only option.
- `ServerAliveInterval 30` keeps an idle connection from being dropped.
- `ForwardX11 yes` is the same as `ssh -X`, for programs that open windows (needs XQuartz on a Mac).

Permissions

SSH refuses to use keys if other people could read or change them. If things aren't working, set these (the same on your laptop and the cluster):

File or folder	Permission	Command
`~/.ssh`	only you can enter (`drwx------`)	`chmod 700 ~/.ssh`
`~/.ssh/id_ed25519` (private key)	only you can read (`-rw-------`)	`chmod 600 ~/.ssh/id_ed25519`
`~/.ssh/authorized_keys`	only you can read/write	`chmod 600 ~/.ssh/authorized_keys`
`~/.ssh/config`	only you can write	`chmod 600 ~/.ssh/config`
`~/.ssh/id_ed25519.pub`	anyone can read (fine)	`chmod 644 ~/.ssh/id_ed25519.pub`

Your home directory itself also must not be writable by others (`chmod go-w ~`).

Troubleshooting

It still asks for my password (and Duo). The cluster didn't accept your key. Run the connection in verbose mode and look at the lines about keys:

```
ssh -v hpcc 2>&1 | grep -i -E "offering|authenticat|identity"
```

- No `Offering public key` line: your laptop isn't using the key. Check the file name in `IdentityFile`, or run `ssh-add ~/.ssh/id_ed25519`.
- `Offering public key` but then it moves on to `password`: the key isn't in the cluster's `authorized_keys` (log in and check with `cat ~/.ssh/authorized_keys`), or the permissions on the cluster are wrong (see the table above).

`Permissions 0644 for '.../id_ed25519' are too open.` Your private key is readable by others; fix it with `chmod 600 ~/.ssh/id_ed25519`.

`WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!` The server's identity doesn't match what is saved in your `known_hosts`. This can happen after the cluster is upgraded, but it can also be an attack. Check the HPCC news or ask support before removing the old entry with `ssh-keygen -R cluster.hpcc.ucr.edu`.

GitHub says `Permission denied (publickey).` Run `ssh -T git@github.com`. Make sure the key on *this* computer (laptop or cluster) has its `.pub` added to your GitHub account, and that your repository uses an SSH URL (`git@github.com:biodataprog/...`), not `https://`. Check with `git remote -v`.

Do and don't

- **Do** use a passphrase, plus `ssh-agent` so you don't have to keep typing it.
- **Do** make a separate key pair for each computer, and remove the public key from `authorized_keys` and GitHub when you stop using a computer.
- **Don't** ever copy, email, upload, or commit a private key (a file *without* `.pub`). If you think it leaked, delete that public key everywhere and make a new pair.
- **Don't** use `>` when adding to `authorized_keys`; it erases the keys already there. Use `>>` or `ssh-copy-id`.

More information

- HPCC: Logging in and SSH keys
- GitHub: Generating a new SSH key and Adding it to your account
- Chapter 4 ("Working with Remote Machines") of Vince Buffalo's *Bioinformatics Data Skills*, free on the UCR network via O'Reilly