

Lecture Agenda and Notes Draft:

Unix Fundamentals: Remote Machine

SSH connections	1
SSH Forwarding	2
Configure the SSH using .ssh/config	3
Copying and Downloading Data	6
Running command on a remote, terminal multiplexing	7
	8

Problem sheet:

Unix Fundamentals: Remote Machine

9

Lecture Agenda and Notes Draft:

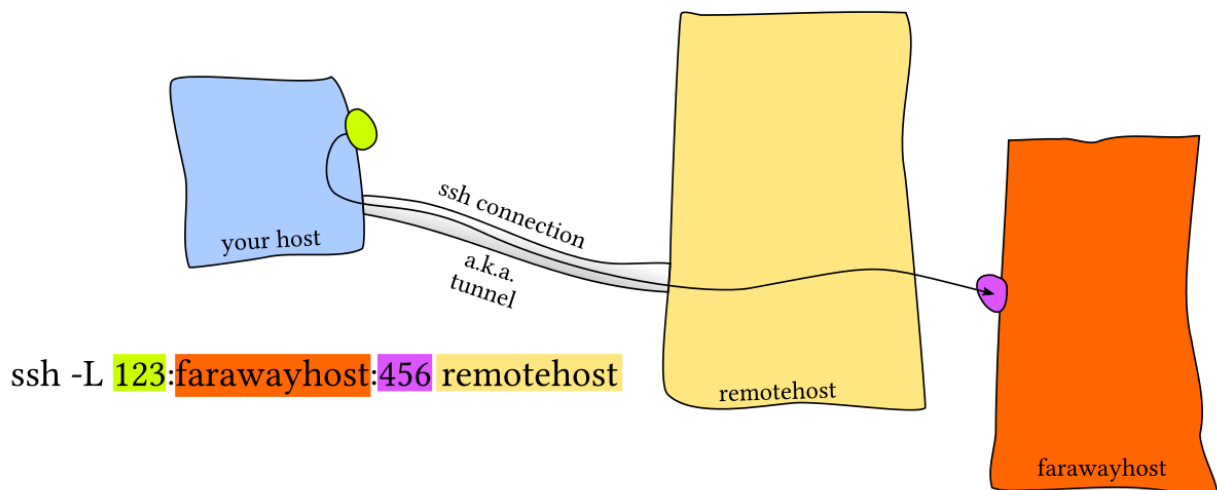
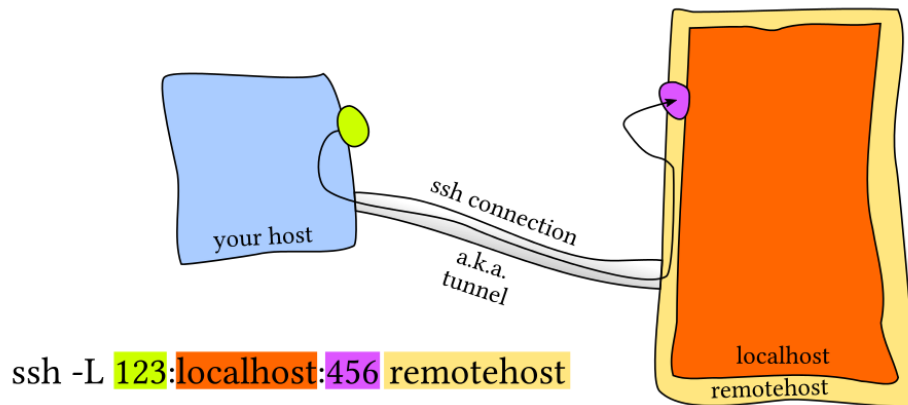
Unix Fundamentals: Remote Machine

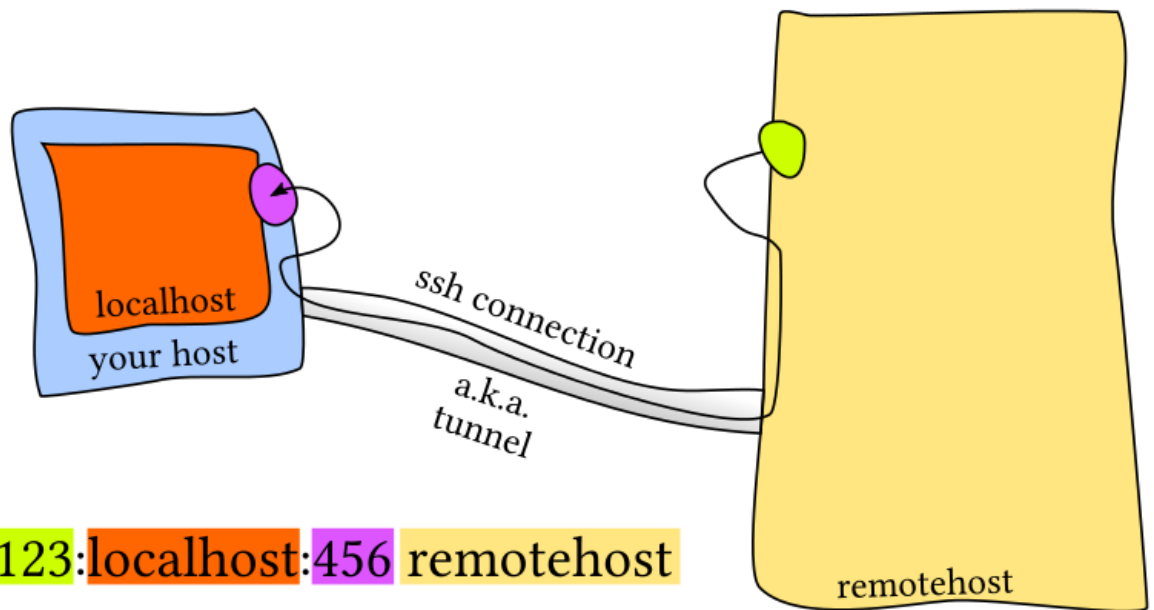
SSH connections

1. Show command `ssh user@machine`
 - a. You are in the shell of a remote machine, you can just type commands and do whatever you want
 - b. execute `logout` to disconnect
2. Sometimes one doesn't need to connect to a shell, you just need to execute a command. For example `ssh user@machine ls`, it'll output the result of the command on the remote
 - a. If you do `ssh user@machine ls | grep -E "Do.*"` then it'll execute first part remotely, second part locally
 - b. If you do `ls | ssh user@machine grep -E "Do.*"` then it'll execute first part locally, send results to a remote, and execute second part remotely
3. You needn't always type your credentials when logging into a remote. It is possible to configure your local `ssh`.
 - a. Run `ls .ssh` in your home directory to see what's in it
 - i. **known_hosts** contains the known keys for remote hosts
 - ii. **config** contains user's own configuration file which, where applicable, overrides the settings in the global client configuration
 - b. Run `ssh-keygen -t rsa -b 4096` to create a public/private key pair that might be used for authentication on a remote server. You can add the **public** key to a server two ways
 - i. Manually append it `cat .ssh/id_rsa.pub | ssh user@remote 'cat >> ~/.ssh/authorized_keys'` (cat writes to stdin of remote, remote append it to the keys file)
 - ii. Use `ssh-copy-id -i .ssh/id_rsa.pub user@remote`

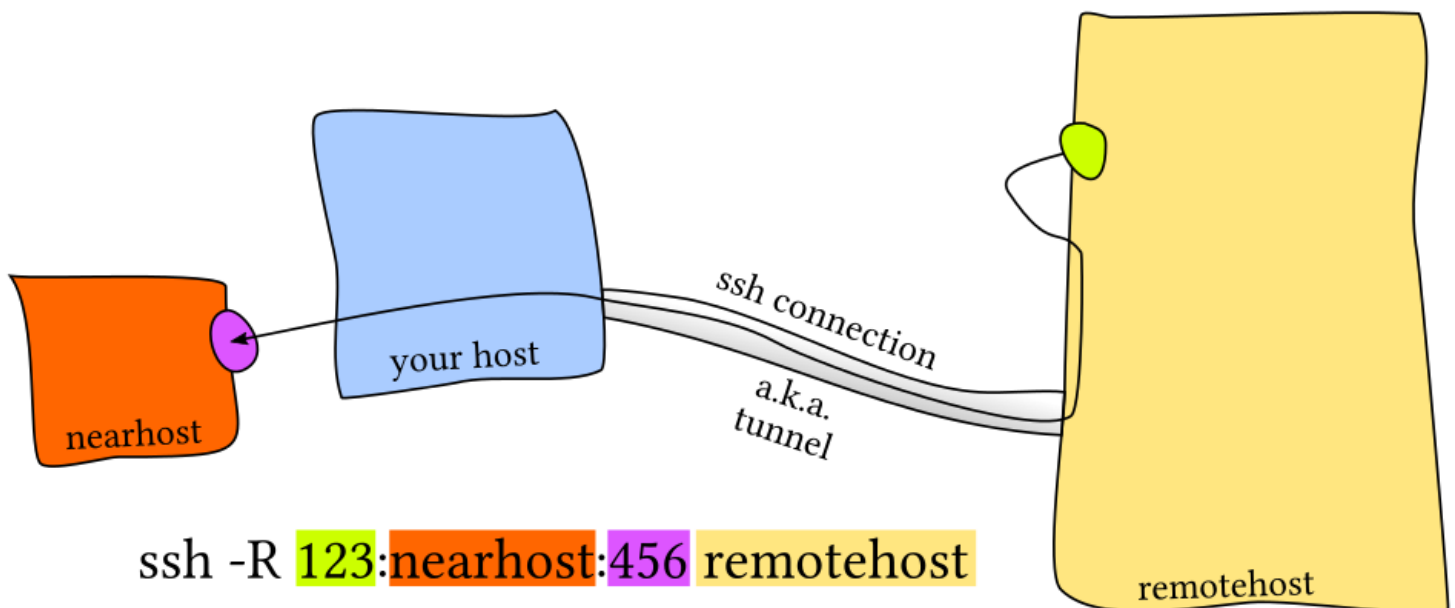
SSH Forwarding

1. Any service that you run on the machine is assigned to a port. For example when you start a jupyter notebook it by default uses port 8888 of your localhost. It is possible to start a service on the remote and access locally using port forwarding.





```
ssh -R 123:localhost:456 remotehost
```



```
ssh -R 123:nearhost:456 remotehost
```

- a. Most of the time, you need to access the remote service on port 456 locally on port 123

run `ssh -L 8888:localhost:9999 remotehost`.

- i. Example 1:

1. connect to a remote with this command,
2. run jupyter notebook on a port 9999,
3. then open locally on your browser localhost:8888

- ii. Example 2

1. connect to a remote with this command,
2. run `nc -l 9999` remotely (start listening on port)
3. run `curl localhost:8888` locally

2. Another type of forwarding is graphics forwarding. This allows you to run GUI based programs on the server and interact with its user interface locally.

- a. For this to work server must have these lines present
X11Forwarding yes; X11DisplayOffset 10
inside file */etc/ssh/sshd_config*
- b. To forward graphics run *ssh -X user@remote*, then run *gedit* (if present) to check

Configure the SSH using .ssh/config

1. One can create aliases for the remote machine available by modifying the .ssh/config file

- a. *Host my_machine*

User my_user # <user> in the ssh <user>@my_machine command

HostName 172.16.174.141 # the name of a remote machine

Port 22 # port to connect via SSH (22 is default)

IdentityFile ~/.ssh/id_rsa # which key to use when authenticate

RemoteForward 9999 localhost:8888 # which port always forward

Configs can also take wildcards

*Host *.skoltech.ru* # connect any machine within the domain with same user

User foobaz

Copying and Downloading Data

1. Similar to how you'd copy on a local with `cp folder1 folder2`, one could do `scp folder1 user@remote:folder2`, to copy local to the remote. This tool is very handy
 - a. add `-r` flag to do it recursively
2. One can use the more reliable tool `rsync`. `rsync` improves upon `scp` by detecting identical files in local and remote and preventing copying them again. It also provides more fine grained control over symlinks, permissions and has extra features like the `--partial` flag that can resume from a previously interrupted copy.
 - a. use `rsync -avP folder1 user@remote:folder2` to copy local folder to the remote showing progress and don't send files if they are already there
3. RClone is used for cloning from drives such as (google drive, dropbox)
4. `wget` is used to fetch data by URL

Running command on a remote, terminal multiplexing

1. If you run a command that takes a really long time on a remote e.g. `sleep 1000`, it'll stop if the connection between your local and remote is lost. Even if you send it to the background using `sleep 1000 &`, it'll have the same effect. To prevent command from stopping while on the remote one can use `nohup` + background, `nohup sleep 1000 &`
 - a. Run `nohup sleep 1000 &`
 - b. Disconnect from the remote
 - c. Connect and run `ps aux | grep sleep`, you'll see that it's still running
2. Or for example you can run interactive commands such as `top`. The `top` will stop when you disconnect. To prevent it one can use `screen` or `tmux`, these both convenient for interactive shell sessions
 - a. Screen
 - i. Run `screen`, then run `top`
 - ii. Press `Ctrl + a` followed by `d` to detach
 - iii. Disconnect from the remote then connect
 - iv. Run `screen -r` to reattach
 - b. More interesting tool is `tmux` (тимукс). In it you can open multiple windows or tabs, split them side to side, detach and attach to a group of these tabs
 - i. To start session run `run tmux`
 - ii. To detach session run `Ctrl+b, d`
 - iii. To attach a session run `tmux a -d`
 - iv. To create a window inside `tmux` run `Ctrl+b, c`; to switch between windows run `Ctrl+b, <some_number>`
 - v. To split current window on panes use
 1. `Ctrl+b, %` to split vertically
 2. `Ctrl+b, "` to split horizontally
 3. To jump between them use `Ctrl+b, <arrows>`