CS6
Practical
System
Skills
Fall 2019 edition
Leonhard Spiegelberg
lspiegel@cs.brown.edu
Recap

Last lecture:

- bash scripting
  - exit codes / status codes / return codes ⇒ 0 success, else failure
  - && and II
  - [ and test
  - bash builtin extension: [[ ... ]]
  - if
  - grouping commands via subshell ( . . . ) and braces { . . . ; }
Fix the following statements! Assume \( x \) and \( y \) are variables.

<table>
<thead>
<tr>
<th>wrong</th>
<th>fixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z=$(x * 3) )</td>
<td></td>
</tr>
<tr>
<td>wrong</td>
<td>fixed</td>
</tr>
<tr>
<td>if ( [x &gt; 10</td>
<td></td>
</tr>
<tr>
<td>wrong</td>
<td>fixed</td>
</tr>
<tr>
<td>echo &quot;x^2 + y^2: <code>x ^ 2 + y ^ 2</code>&quot;</td>
<td></td>
</tr>
<tr>
<td>wrong</td>
<td>fixed</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>$z=$(x * 3)</td>
<td>$z=$((x*3))</td>
</tr>
<tr>
<td>if [x &gt; 10</td>
<td></td>
</tr>
<tr>
<td>echo &quot;x^2 + y^2: <code>x ^ 2 + y ^ 2</code>&quot;</td>
<td>echo &quot;x^2 + y^2: $((x ** 2 + y ** 2 ))&quot;</td>
</tr>
</tbody>
</table>
08.01 Basic networking

Networking usually follows the pattern of a client connecting to a server and performing a request which yields a response.
08.01 Basic networking

⇒ to connect to a server, we need to know its address.

⇒ in a network, each device is assigned an IP (Internet Protocol) address. Two flavours:

  ⇒ IPv4: 192.168.0.1 (32 bit integers, 4 8-bit segments)
  ⇒ IPv6: fe80::c8c:de7c:82dd:6012 (128bit, 8 16-bit segments)

⇒ a machine is also called a host, which has a hostname

⇒ you can use hostname to get the hostname of your machine
08.01 Basic networking

⇒ one *host* communicates with another *host* over a connection.

⇒ the outlet (or endpoint) over which the communication occurs is called a *socket*.

⇒ On a machine there are $2^{16}$ sockets available, identified by a 16 bit unsigned integer. Each socket binds to a unique *port* numbered 0-65536.

⇒ port numbers $< 1024$ are reserved.
A socket is an end-point of a two-way communication link of two programs running on a network. Each socket is bound to a port number 0-65536.
08.01 Sockets and IP addresses

⇒ to specify a connection we need two IP addresses and one port

Client
192.168.0.20

communicate via port 80

Server
192.168.0.30
08.01 Communication layers

⇒ a protocol defines how two hosts/devices communicate

⇒ OSI = Open Systems Interconnection model is a model to allow different systems to communicate along clearly defined abstractions and standards

⇒ different (abstraction) layers for communication with each of them having different protocols

⇒ in CS6 we only care about host layers

⇒ more on the OSI model: https://www.cloudflare.com/learning/ddos/glossary/open-systems-interconnection-model-osi/

⇒ more on networks in CS168
08.01 Labels for IPs

⇒ IPs are hard to remember and assignment of IPs frequently changes

⇒ there are multiple ways to assign a label to an IP

⇒ depending where the machine we want to connect to is located, we can use different options to name it:

   → hostname i.e. a tag to a computer in a network
   → domain name i.e. a tag to use with a service which provides a final tag or address
08.01 Defining hostnames

⇒ hostname reveals the name under which the current machine can be reached

⇒ we can manually tag an IP, by editing /etc/hosts (requires root access)

/etc/hosts

```plaintext
1 ##
2 # Host Database
3 #
4 # localhost is used to configure the loopback interface
5 # when the system is booting. Do not change this entry.
6 ##
7 127.0.0.1 localhost
8 ::1 localhost
```
08.01 Looking up IPs via URI resolution

⇒ resources can be identified via a URI=Uniform Resource Identifier

Generic syntax:

```
URI = scheme: //[authority]path[?query][#fragment]
```

The authority itself can be split into

```
authority = [userinfo@]host[ :port]
```

**Note:** `path` starts with `/`, which is considered part of the path.
08.01 URLs are URIs

⇒ URL = Uniform Resource Locator
   (often referred to as web address) is used to reference a web resource

https://cs.brown.edu:80/courses/cs0060/index.html
DNS = Domain Name System

⇒ translates URIs (incl. hostnames) through DNS servers to IP addresses
08.01 hostnames to IP

⇒ `getent hosts unix.stackexchange.com` to list addresses under which `unix.stackexchange.com` can be reached

⇒ to restrict to IPv4 only, use `getent ahostsv4 hostname`

⇒ *NIX tries to resolve hostname via multiple services, thus multiple IPs may be available for one URI.

getent works under Linux, use `dns-sd -q hostname` under Mac OS X
How can we access a remote machine?
08.02 Working remotely - historic commands

⇒ as part of BSD, programs rlogin, rsh, rexec were shipped

  rlogin  allows you to login into a remote machine
  rsh  remote shell, allows you to open a shell without login
to execute arbitrary commands
  rexec  Like rsh but with login, reads username and
password (unencrypted) from a socket

⇒ Problem: All these tools send user passwords over the network in a
clear format, without any encryption. This is a security risk!

⇒ rlogin is the worst, by relying on IP addresses for authentication; but
it's easy to fake an IP address and take over a remote machine!
How to encrypt data, passwords, user names to securely work with a remote machine?
Symmetric encryption:
same key is used for both encryption and decryption
Some widely used symmetric encryption algorithms are: Blowfish, AES, RC4, DES, RC5, and RC6

⇒ widely used is AES, which can be used with 3 different key sizes: 128, 192 or 256 bit

⇒ The more bits the key has the better the encryption; but the slower encryption/decryption

We can use openssl to encrypt/decrypt a file!
08.03 AES-128 via openssl

⇒ to encrypt a file use

Encrypt:

openssl aes-128-cbc -e -pass pass:secret \  
-in file_to_encrypt.txt -out encrypted.txt

Decrypt:

openssl aes-128-cbc -d -pass pass:secret \  
in encrypted.txt -out decrypted.txt

⇒ openssl provides many more features, i.e. man openssl or openssl help
Remaining problem:
How to exchange the key?
08.04 Asymmetric/public key cryptography

Generate two keys: one public key and one private key

⇒ share and use public key to encrypt message, but **only** holder of private key can decrypt message.

Bob, Stop trying to make fetch happen. - Alice

Bob's Public Key

PIQ6NZOKW CXSL03zta+ soRTuwJ/7J0 Q7gzwyJBuy CYBn

Encrypt

ciphertext

Bob's Private Key

Bob, Stop trying to make fetch happen. - Alice

Decrypt

plaintext

plaintext

Keys are different but mathematically linked
How to exchange a key?

Diffie-Hellman-Merkle key exchange

Alice

private R

private R + common Y = public RY

private R + public GY = private RGY

Bob

public G

public G + common Y = private G

public G + public RY = private RGY

⇒ allows you to create a shared, private key! Details in a cryptography class, e.g. CS151
08.04 Diffie-Hellman-Merkle exchange

⇒ can be used to share a secret key, which then may be used for following symmetric encryption

⇒ Problem:
Man-in-the-middle attack possible because no authentication that public keys are from actual Alice/Bob respectively.
RSA is a true public cryptography algorithm named after Rivest-Shamir-Adleman
RSA vs. Diffie-Hellman-Merkle

⇒ RSA can be used for both exchanging a key OR direct, asymmetric encryption.

⇒ Also DHM can be used for both exchanging a key OR direct encryption

⇒ they use different underlying principles and are vulnerable to different attacks

⇒ symmetric cryptography is usually faster than asymmetric cryptography

⇒ Details in Cryptography class
08.04 Public key cryptography

Summary:

Generate a key pair, **ONLY** share the public key.
**NEVER** share the private key.

⇒ for additional security, private key is often protected by a passphrase. I.e. the private key for asymmetric encryption is encrypted using a symmetric encryption (per default AES-128).

⇒ Advantage: If someone gains access to your system, private key still somehow encrypted.
Practical public key cryptography...
...thanks to SSH!
SSH = Secure Shell

⇒ invented 1995 at Helsinki University of Technology, Finland
⇒ cryptographic network protocol to allow safe remote login
⇒ replaced previously used standards such as rlogin, rsh, rexec and telnet
⇒ defacto standard way to work with other machines over a network today
⇒ uses port 22 per default
08.05 SSH protocol

⇒ ssh handles the set up and generation of an encrypted TCP connection
⇒ allows to login securely remotely (ssh)
⇒ allows to copy files securely (scp)
08.05 SSH programs

⇒ there are two programs:

Client: ssh
Server: sshd ⇐ runs in the background

⇒ if sshd is not running, you can not login

⇒ different implementations for ssh/sshd
   most popular one: OpenSSH
08.05 SSH authentication options

⇒ SSH provides 4 different authentication methods

1. Password

2. Public/private keypair ⇐ this is the one you should use

3. Host-based authentication

4. Kerberos
08.05 SSH - Password authentication

tux@server $: ssh remote-machine
password:
remote-machine $:
Using public/key cryptography to login:

1. Generate keys via `ssh-keygen` (generates private and public key)
2. Copy public key to remote machine and append it to `~/.ssh/authorized_keys`
3. login via `ssh -i <private-key-file> remote-machine`
08.05 Step1 - Key generation

⇒ you can generate a key pair using `ssh-keygen`

⇒ keys should be stored by default in `~/.ssh/id_rsa` (private) and `~/.ssh/id_rsa.pub` (public)

⇒ ssh uses default names, i.e. if `id_rsa.pub` and `id_rsa` exist in `~/.ssh`, you can login without specifying a key explicitly.

⇒ you can protect your key with a passphrase (recommended!)
**08.05 Step1 - Key generation**

**ssh-keygen**

<table>
<thead>
<tr>
<th>useful options</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-t rsa</code></td>
<td>select rsa for key generation</td>
</tr>
<tr>
<td><code>-b bits</code></td>
<td>how many bits to use for key, anything larger than 1024 is good</td>
</tr>
<tr>
<td><code>-N new_passphrase</code></td>
<td>specify passphrase on the fly to protect key</td>
</tr>
<tr>
<td><code>-f output_keyfile</code></td>
<td>creates private key under output_keyfile and public key under output_keyfile.pub</td>
</tr>
</tbody>
</table>
08.05 Step1 - Key generation

Example:

```
tux@server:~$ ssh-keygen -b 2048 -t rsa -f tuxkey
Generating public/private rsa key pair.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in tuxkey.
Your public key has been saved in tuxkey.pub.
The key fingerprint is:
SHA256:RozagpJ2mW8gki9MvVHTypZldp1DsIkof1vx3L8rLcc tux@server
The key's randomart image is:
+---[RSA 2048]----+
|    ...    |
|    oo. = .   |
|    . +.*o= + |
|  o..BoB.. + o  |
|+=+.Bo*..S. o   |
|==oo *...o       |
|.... o .   o.   |
|   . . o E.|    |
|     +o.    |
+-----[SHA256]----+
```
08.05 Key files

private key

-----BEGIN RSA PRIVATE KEY-----
MIIeQIBAAKCAQEAYy7wcb648cQIZQZjZ+VZkVZzPz4A4b4W4x7P7jDXL7XtZSYkYe
E5k7cE5V4CjEhA1W7jE3w+10imIny/NyQ/jvBnWjK87QPyhY4C5qkurAhypY3+
HpZ1hDH2Mt7v7m7w9ug73yr69bCDA56b3793170t5L7s/Q1/C2/NV4z
PZL7Xa0NstjM0PZ7Lke3eoM7p8nJ3nAw2V2U4DEOcERsTsd+H0ph351p2vL
WLCUt/w/4jK98s1cDPoayj4KjF6U14w0U4XJXOYFJK7k6C8yPxBqKdKcvce7ocX
2Tgwh4w03Z62M1zuQo6qEPA47/6Taej40xGkEQsedIAQABa01AbHszS5g20MzufG5G
Hq99dcu73PvYK7P77Wnt95D5Kcb5Ba0Hd7j10IRmbvXz0upAYV1807u9b4Ww7R+v+
Bz1zBm7j4CRIKXlvlg8xRvN/s5vb/eiq7/7XP45MT2Hn3sLzD3ZJRyFjswQxu
CI0vz2v+wv1qsMsGqulr5g73hj/jUp8l2kFrLsS8/Bh5xzcCpfvO4K1v1wRQSw
Amrjrc3wlsSp9fNFRSR5061jVf7C7Kr3e74W57dmasip6cxZyXhkezFFFFSHRdAS
hyZEPcP16l4MNZHyInxjwzZ68bY8YD5ZirzfPp1l4jE1bkrRygyvzhcDd7v7b/hkfr/
04FdBgAgCyyEAg5d7auj/0YI6CmmY7WCI7E5bn6+M+Nwu+Hag3UE7A3pLjnt/qPxl7q
WiffF8K7kJp1XeJ6H4KRQ8RbiKtikswSnc99dcbF+6rcRnUny8eLrt8rsUPYQh
Hx8PqLR6cN6s81u3Qu+i1Q5uNeaGTVxryxqDcXwWIdzLk111ncn2g6eCyEA4uc
roN9n+BQ+yOf/Wnc8K1Uqb18zN8eJ7eIMFUEquX4xN8DcZC0T6bTBE13K+MyMtt
S8uqupa6FgbDxMr6zsdw3gIeegdptqC4xuQqbj5s/3yZfeZbJhFRlP6wDHn7r
umyK+kAC8rfu+I11NhW4R8kBYo8gAtSrAgjK8CgyYEAgqu6Gmp61fFSYEmIA6w
zhCOk1sA5jcQpwmmNt1323Ovldc0qlm8jfmQ4iuTdiqMy2p/fAzbX+BaeEmCbu6
IpeFvQvVR6y6CvJWTjWd6wmbCJBF69M7JnA1yQZ057x6ju3L3BafkWw4l1u1jMADPw
udI2WJ2q6CipMDedh/CggEcyEAYuqVqyrA8C0dr5NOVQjHox3Y1HgMw1kdMNC
ESehAc8x9BPw10rH+u5coqbd2ULJyAyEH60McGF9hdV161f4j1GgsqkuhxzH0+apH
QmWx1szhW+quOBU1pves2d37bYwQaj1FBFxta1wpgkswksk52XbhKMdy92dCQWL
r9ux1EYAMcKvO60C9O0NtFdzqazYWEsbaFMae/T6R6Pm9frjX48BvyJ0szk
ptFANsnbZsdCpCJHof/FttnIpsFM80PwXmp/FLv1t6yno1aFaJUsKlqceOu5
CeIt0v1g1fjCdc1nATy3gurRC7qYnW90kHj+c95WdP8GzUryAn3u=
-----END RSA PRIVATE KEY-----

public key

ssh-rsa AAAAB3NzaC1yc2EAAADAQABAAABAABDQKs/Bxt3jxw4hBk29kyRSjM+UBHiYFjs/GMPUtdC3N1FiQB
4T/mTtwQE1Ti0IdsQCjbB2MTfCltXRysidX+dhD+Mrse
yYnxBuEnKFL1la4A+fiJf4yNycUNscwa1jubvC46D
btCCIGbpAIMUw5obrvcxAvx3vSSx6Cwwv+ULb8xXgi
kt1Bdo41Ky0gyg/vMsp4GypsGCh0cCttta7ZyTmQI
RGhNJ0r4c6mHmGmx87Ys3/An/gr3ySII05889rBZ
gl/RTXhY5TiNc5g UmQbuRzzE/K8GpEpJyty/S6wJfaj
0b7Ac7cHMzXNjndQkQ8DhP/pMB6P7G4Qv
mux@server
08.05 Step2 - copying ssh keys

⇒ append them to \texttt{~user/.ssh/authorized_keys} on the machine where you want to login to authorize login as \texttt{user}

⇒ Tip: use >> to append to \texttt{~user/.ssh/authorized_keys}

⇒ On Linux, there is also a command to copy keys to a remote machine: \texttt{ssh-copy-id}

⇒ Example: \texttt{ssh-copy-id -i ~/.ssh/id_rsa user@host}
08.05 Step3 - Logging in

To login into the remote machine use

\[ \text{ssh -i <private-key-file> user@host} \]

or

\[ \text{ssh -i <private-key-file> host -l user} \]

⇒ can be cumbersome to write login details always, to save
default edit \(~/.ssh/config\)!
08.06 Configuration via ~/.ssh/config

⇒ can store details for a host there

```
~/.ssh/config

Host cs6demo
    HostName 18.206.152.69
    User tux
    IdentityFile ~/.ssh/tux.key
```

⇒ i.e. after adding tux.key.pub tp ~tux/.ssh/authorized_keys on remote machine, login is as simple as ssh cs6demo!
⇒ SSH uses a (RSA) fingerprint to verify the identity of the server/remote. When connecting the first time, usually a prompt ask whether to accept the server's fingerprint.

⇒ If ssh warns about mismatch of the fingerprint, the following scenarios might have happened:

1. the key used to generate the fingerprint changed on the server (SSH or OS update)
2. hostname or IP belongs to different server now
3. Malicious man-in-the-middle attack
use `ssh-keygen -R hostname` to remove entry for hostname from known_hosts

you can add a fingerprint using `ssh-keyscan` via

```
ssh-keyscan -H hostname >> ~/.ssh/known_hosts
```
~/.ssh/known_hosts
⇒ used to verify the identity of remote hosts

~/.ssh/authorized_keys
⇒ keys authorized to login on this machine

~/.ssh/config
⇒ login configuration
use `\texttt{-v}` to see what's happening under the hood!

tux@server:~$ ssh -v user@ssh.cs.brown.edu
OpenSSH_7.6p1 Ubuntu-4ubuntu0.3, OpenSSL 1.0.2n 7 Dec 2017
debug1: Reading configuration data /etc/ssh/ssh_config
debug1: /etc/ssh/ssh_config line 19: Applying options for *
debug1: Connecting to ssh.cs.brown.edu [128.148.31.18] port 22.
debug1: Connection established.
debug1: SELinux support disabled
debug1: key_load_public: No such file or directory
debug1: identity file /home/tux/.ssh/id_rsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/tux/.ssh/id_rsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/tux/.ssh/id_dsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/tux/.ssh/id_dsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/tux/.ssh/id_ecdsa type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/tux/.ssh/id_ecdsa-cert type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/tux/.ssh/id_ed25519 type -1
debug1: key_load_public: No such file or directory
debug1: identity file /home/tux/.ssh/id_ed25519-cert type -1
debug1: Local version string SSH-2.0-OpenSSH_7.6p1 Ubuntu-4ubuntu0.3
debug1: Remote protocol version 2.0, remote software version OpenSSH_7.9p1 Debian-10
debug1: match: OpenSSH_7.9p1 Debian-10 pat OpenSSH* compat 0x04000000
debug1: Authenticating to ssh.cs.brown.edu:22 as 'lspiegel'
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kek: algorithm: curve25519-sha256
debug1: kek: host key algorithm: rsa-sha2-512
debug1: kek: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: kek: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLY
debug1: Server host key: ssh-rsa SHA256:P4ZsteVHDJ1nFY6UHFh1VTkRqRJxvBmt16IhLS+EqoE
The authenticity of host `ssh.cs.brown.edu (128.148.31.18)` can't be established.
RSA key fingerprint is SHA256:P4ZsteVHDJ1nFY6UHFh1VTkRqRJxvBmt16IhLS+EqoE.
Are you sure you want to continue connecting (yes/no)?
ssh can be used to run arbitrary commands remotely

⇒ ssh cmd [param1 ...]

Example:

ssh cs6server ls /home/tux

⇒ helpful to install things remotely, execute scripts, ...

⇒ can use this with all the other bash tools!
08.07 More on SSH

⇒ X Server provides (one way for a ) graphical user interface (GUI) on Linux

⇒ we can start programs remotely and display their GUI via X-Server forwarding ssh -X

Example:

ssh cs6server firefox library.brown.edu

⇒ Further application: SSH can be used to tunnel ports, i.e. exposing a secured port of a remote locally.
08.08 Transferring files: scp

scp [-r] [-i identity_file] [[user@]host1:]file1 ... [[user@]host2:]file2

⇒ there are many more options, however -i to specify private key is the most useful.

⇒ scp copies files between hosts using SSH protocol

Example:

scp -r folder tux@cs6demo:/home/tux/

copies folder recursively to /home/tux/.
08.08 scp examples

local file to remote host

scp file.txt user@host:/remote/directory

remote file to local cwd

scp user@host:/remote/directory/file.txt .

remote host to remote host

scp tux@from_host:/home/tux/file.txt sealion@to_host:/home/sealion/
08.08 scp limits

⇒ *scp* can be slow for large files as it performs a plain copy, i.e. read/writes all files and encrypts data via SSH

⇒ *use scp -c cipher* for faster encryption by selecting a fast cipher explicitly (*ssh -Q cipher* to list available algorithms)

⇒ *scp* is not able to resume copying, show progress, ...
Is there another tool?
08.08 rsync

rsync = remote sync

⇒ can be used to sync directories, download files, resume transfers, show progress, ...

⇒ similar syntax to scp, use -e ssh to use SSH as protocol

⇒ more powerful options, e.g

--progress shows progress on transfers
--include '*.csv' include only csv files
--exclude '*.tmp' exclude tmp files
--max-size='1m' maximum size of files to be transferred
--dry-run perform dry-run, i.e. no materialization
End of lecture.

Next class: Thu, 4pm-5:20pm @ CIT 477