CS2043 - Unix Tools & Scripting
Cornell University, Spring 2014

Instructor: Bruno Abrahao

January 29, 2014

---

1 Slides evolved from previous versions by Hussam Abu-Libdeh and David Slater
Accessing Remote Resources
You can use “secure shell” (ssh) to connect to a remote machine.

```
ssh [username@]<remote machine name or IP address>
```

- If the username is omitted, local username will be used.
- Remote machine has to be configured to accept ssh connections:
  - ssh daemon (service) has to be running and listening on an open port (by default 22)
**Executing remote commands**

`ssh` can be used to execute commands on the remote machine.

**Example**

```bash
ssh ha232@csug01.csuglab.cornell.edu ls
```

This will execute `ls` on `csug01.csuglab.cornell.edu` and output the result to the screen before `ssh` terminates the connection.

- The `-f` flag puts `ssh` into the background before executing the remote command.
- The `-Y` or `-X` (interactive) flags forward X11 (graphical user interface) to the local machine.

**Run firefox on the remote machine**

```bash
ssh -Y ha232@csug01.csuglab.cornell.edu firefox
```
Identity files

- An identity file authenticates with the remote machine instead of using your username/password.
- Allows you to authenticate yourself with a “pass phrase” (which could be empty).
- Consists typically of a pair of public/private keys used for asymmetric key cryptography (e.g., RSA).
If you don’t want to type your password every time.

**create identity files**

```
ssh-keygen -t dsa
```

Append the generated public key file (by default `~/.ssh/id_rsa.pub`) to the `~/.ssh/authorized_keys` file on the remote machine.
If you don’t want to set the corresponding flags every time, use the ssh config file `~/.ssh/config`.

### Sample config

```
host office
  hostname mishmish.cs.cornell.edu

host tiger
  hostname tiger.cs.cornell.edu
  user Alice
  ForwardX11 yes
  IdentityFile ~/.ssh/id_rsa
```

Here, `ssh office` connects to `mishmish.cs.cornell.edu` and `ssh tiger` connects to `tiger.cs.cornell.edu` with username Alice and identity `~/.ssh/id_rsa` and enable X11 forwarding.
Secure copy

**scp**

- Copy files securely over a network using an encrypted **ssh** transport.

**copy file to remote machine**

```
scp file [username]@remote_machine:
```

**copy file from remote machine**

```
scp [username]@remote_machine:file .
```

- The `:` is necessary after the remote machine name. A path on the remote machine starting from the user’s home directory can be specified after the colon `:`.

**copy directories using the `-r` flag**

```
scp -r pics/ remote_machine:
```
Other remote data transfer commands

wget

wget [OPTIONS] [URL...]

Download a file from a remote location over HTTP. Popular options:

- `-r`: recursive
- `-l[number]`: how many levels to descend when following links
- `-c`: continue a partial download

curl

curl [OPTIONS] [URL...]

Transfer data from/to web servers.

For more info on these commands, consult the man pages.
Processes and Jobs
### Definition

A **process** is an instance of a running program

- More specific than "a program" because it's being executed.
- More specific than "a running program" because the same program can be run multiple times simultaneously or use multiple processes

### Example:

Many users could be simultaneously running program on the same computer. In this case, each *instance* of running program is a separate process.
Process Identification

How do we tell one process from another?

- Each process is assigned a unique "Process ID" (or PID) when it is created
- These PIDs are used to differentiate between separate instances of the same program

How do we find out which processes are running, and with which PIDs?

The **Process Snapshot Command**

`ps [options]`

- Reports a snapshot of the current running processes, including PIDs
By default, `ps` is not all that useful because it only lists processes started by the user in the current terminal. Instead...

**ps Options**

- `ps -e` – Lists every process currently running on the system
- `ps -ely` – Gives more info about your processes than you’ll ever need
- `ps -u username` – Lists all processes for user username.

**NOTE:** Options for BSD are different!

To see information about a specific process, pipe through `grep`.

For example,

```
ps -e | grep firefox
```

shows us information about all firefox processes.
Remember that although UNIX seems to run tens or hundreds of processes at once, one CPU can only run one process at a time.

Quick switching back and forth between processes makes it seem as though they are all running simultaneously.
Suppose you want to run some long running scientific calculation that might take days and consume 100% of the CPU on some server. Other users are going to see your username and think you’re impolite.

Is there a way to tell the server to give your process less priority with CPU time?

Is there a way to give high priority for critical tasks??

UNIX Developers saw this type of situation coming - each process is given a priority value when it starts.
Being nice to others!

Start a process with a non-default priority:

### The nice command

`nice [options] command`

- Runs command with a specified "niceness value" (default: 10)
- Niceness values range from -20 (highest priority) and 19 (lowest priority)
- Only root can give a process a negative niceness value!
- Commands run without nice have priority 0.

### Example:

`nice -n 10 azureus`

- Keeps torrent downloads from hogging all our CPU time!
Adjusting Priority

Adjust the niceness of a running process:

The *renice* command

renice <priority> -p <PID>

- Changes the niceness of the indicated process to <priority>
- Again, only root can go below 0!
- Can only renice processes YOU started!

Example:

renice 5 -p 10275

- Sets the niceness of the process with PID 10275 to 5 (slightly lower priority than default)

renice 19 -u <username>

- renice all user’s processes to 19
To end a process running in the foreground simply hit Ctrl + C

- What about a background process that stops working?
- What is the UNIX version of CTRL + ALT + DELETE?
The kill command

**kill**

```
kill [-signal] <PID>
```

- Sends the specified signal to the process
- By default, terminates execution

So to terminate a process:

- Look up the process’s PID with `ps`
- Use that PID to `kill` the process
Feeling extra vengeful?

```
killall
killall [-signal] <name>
```

Kill processes by name

**Example**

```
killall firefox
```
Useful Kill Signals

Signal used with `kill` can either be specified by their names or numerical values.

- **TERM** or 15: Terminates execution (default)
- **HUP** or 1: Hang-up (restarts the program)
- **KILL** or 9: Like bleach, can kill anything

Generally speaking, the default `TERM` will get the job done.

**Example:**

- `kill 9000`: terminates process 9000
- `kill -9 3200`: force kill on PID 3200
- `kill -HUP 12118`: Restarts 12118 (handy for servers & daemons)
top is a useful little program that lists and dynamically updates information about running programs. It also allows the user to kill and renice other processes.

**top**

```
top [-options]
```

- Lists processes by default by percentage of CPU usage
- Customizable display, hit `h` for a list options
- `u` - show specified user only
- Can manipulate tasks: `’k’` kill; `’r’` renice
Job control is a built-in feature of most shells, allowing the user to pause and resume tasks, as well as run them in the background (so that the shell is usable while it executes!)

**Jobs**

A Job is a process running under the influence of a job control facility.
How does this help?

Lets use the `ping` command as an example.

**Ping**

```
ping <server>
```

- Measures the network response time (or network latency) to a remote server.
- Sends short bursts to the server, then measures the time until they are returned.
- Can be a good way to check your network connection

Try pinging a reliable location, like `google.com`

**Example:**

```
ping google.com
```
Why We Need Job Control

As long as ping runs, we lose control of our shell. This happens with many applications which run either indefinitely or for long periods:

- Moving large quantities of files
- Compiling source code
- Playing multimedia
- Doing Scientific Computing

Example:

mpg123 song.mp3
Starting a Job in the background

To run a job in the background, we will use a new command-line operator:

&

<command> [arguments] &
- Runs the specified command as a background job
- Unless told otherwise, will send output to the terminal!

Since `cat` runs indefinitely with no arguments, this will illustrate our point:

Example:

```
cat &
```
- Try it without the &!
Backgrounding a Running Job

What if we start a process normally and it’s taking too long?

Pausing a Job

Press CTRL + Z to suspend a running process!

- When we do this, the shell tells us the suspended job’s JOB ID
- This Job ID is used like a process’s PID
- Once we have a process suspended, we can tell it to continue in the background...
The Background Command

**bg’s Usage**

```
bg <Job ID>
```

- Resumes a suspended job in the background
- Without a Job ID resumes the last job placed in the background

how do we find these Job IDs?

**the Job Table**

```
jobs
```

- Prints currently running, suspended, or recently stopped jobs
- Prints jobs with their Job IDs
What if we want to resume a job in the foreground?

**fg’s usage**

```
fg <Job ID>
```

- Resume a paused job in the foreground
- Again without a Job ID resumes the last command placed in the background
Killing Jobs

kill

kill %<Job ID>

Alternatively, you can also either foreground it and the hit CTRL+C, or you can use the kill command with the PID