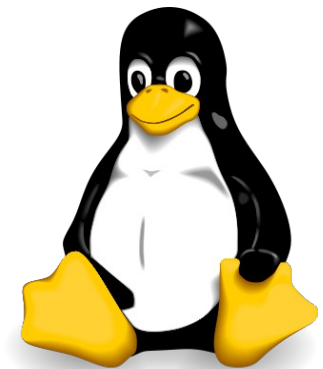


# Linux Misc.

---

make clean



# LWP

Perl has the ‘backticks’ feature

```
$x = `wget google.com`;
```

Can also use (platform independent) LWP

```
$x = new LWP::Simple();  
print $x->get(“google.com”);
```

# Full Mechanization

Any language can download web pages

Not every language comes with support for fully emulating a web browser

Enter: WWW::Mechanize

# Sample Mechanization

```
use WWW::Mechanize;
my $mech = WWW::Mechanize->new();

$mech->get( $url );

$mech->follow_link( n => 3 );
$mech->follow_link( text_regex => qr/download this/i );
$mech->follow_link( url => 'http://host.com/index.html' );

$mech->submit_form(
    form_number => 3,
    fields      => {
        username => 'mungo',
        password  => 'lost-and-alone',
    }
);

$mech->submit_form(
    form_name => 'search',
    fields    => { query => 'pot of gold', },
    button    => 'Search Now'
);
```

# A Custom DynDNS

DNS – Domain Name Servers

Responsible for translating name “google.com” into an IP

A record on a server, needs to be updated when a server’s IP changes

Usually providers charge for DynDNS – How could we do this ourselves?

# Step 1 – Identify how to modify DNS Record

Typically your Domain Name provider will have a web form to help you modify your DNS records.

Choose another domain: bluesata.com	
Start of Authority (SOA)	Serial: 2008101601 Refresh: 10800 (3 hours) Retry: 3600 (1 hour) Expire: 604800 (1 week) TTL (time to live): 86400 (1 day)
Name Server (NS) <b>If you did not register this domain on PowWeb, please contact your registrar and set the Name Servers to the records indicated here.</b>	ns1.powweb.com ns2.powweb.com
	Hostname: IP: email: 65.254.250.40 Hostname: IP: webmail: 65.254.250.40 Hostname: IP: mail: 65.254.250.101 Hostname: IP: smtp: 65.254.250.101 Hostname: IP: imap: 65.254.250.101 Hostname: IP: pop: 65.254.250.101

# Declaring a variable

**FOO=BAR AND RAB**

Anything to the right of the = up and to a newline is added to the variable to the left of the =

Variables are all strings

## Step 2 – Identify when your IP Changes

I use a third party server that I have access to, to keep track of my IP over time

Every night ask it “Has my IP Changed since yesterday?”

It can do an easy lookup to compare the IP now to what it wrote down yesterday.

If it responds with yes then we need to change our DNS Record.

## Step 3 – Update the Web Form

At this point we can email the administrator that our IP has changed, but that's lame. The Admin still has to do stuff.

We can change the record with `WWW::Mechanize!`

Use `WWW::Mechanize` to navigate to the form and fill in the data for us!

## Step 4: Done

That's it! Our server will now know when its IP changes and use Perl with WWW::Mechanize to update our DNS Record!

One of hundreds of thousands of ways to use scripting to make system administration easier.

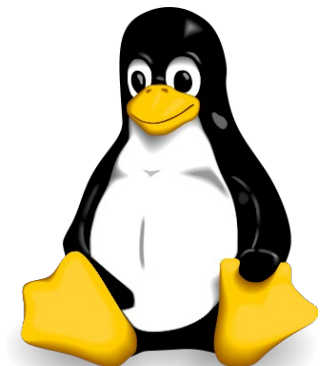
# Makefiles

---

Used to help compile lots of code at once.

**SIMPLE** programming language syntax.

More of a config file than a programming language.

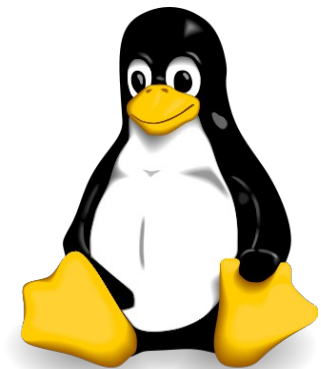


# Declaring a variable

## FOO=BAR AND RAB

Anything to the right of the = up and to a newline is added to the variable to the left of the =

Variables are all strings

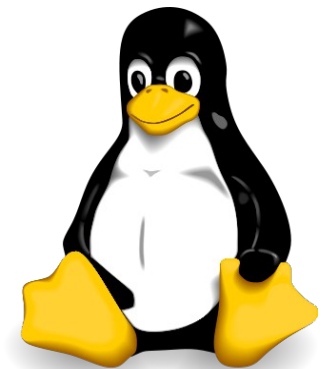


# Accessing a variable

---

`$FOO`

Similar to bash, \$ for accessing



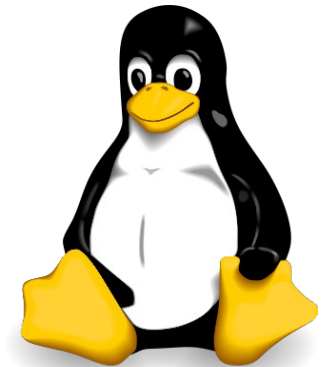
# Rules

---

TARGET : DEPENDENCIES

<TAB>      COMMANDS

The idea: You want TARGET to always be up to date.  
TARGET comes from DEPENDENCIES. If you  
find out TARGET is out of date by checking  
DEPENDENCIES then do COMMAND.



# The End

---

Thats it! Gawsh, Makefiles are simple!

Not really – theres much more 'voodoo' Makefiles can do.

For the most part this covers all of the key functionality of make.

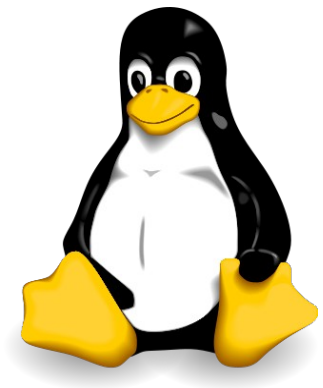
Makefiles often considered 'black magic' in big projects because they can do crazy things.



# Version Control

---

Don't you just love the “VC”s ?



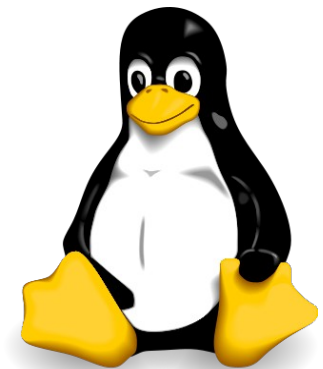
# Purpose

---

Keeping track of how a file changes over time

Stores incremental diffs of all files under “version control”

Lots of files actually!



# History

Linux, arguably the LARGEST distributed project in the world used to send TARBALLS back and fourth to control versioning

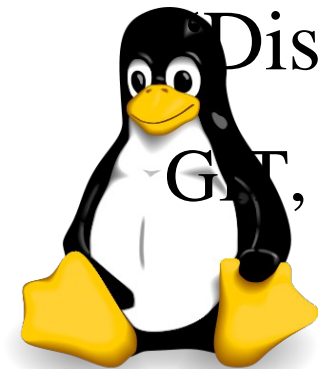
Then they moved to the “Central Repository Model”

CVS, SVN

Then we became “Version Control 2.0” with the

“Distributed Model”

Git, bazaar, mercurial



# Tarballs

---

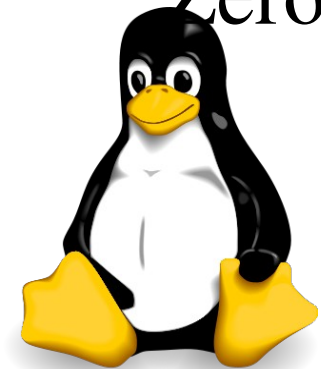
Tar ball is just a collection of files

Intact, full files. Maybe compressed, but still the whole file.

THE ENTIRE FILE. No extra information.

None.

Zero.



# Centralized

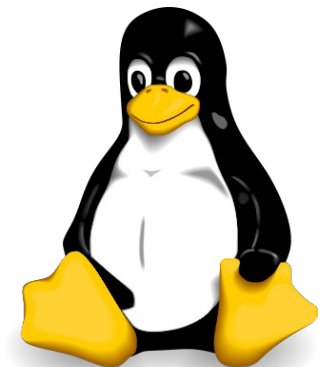
---

The code is in one central “Repository”

Coders “Check out” revisions, edit, then “check in” (or “commit”)

Everybody shares one central repository

Must have access to repo to commit (need internet)



# Distributed

Everybody gets their own repo! Free beer too!

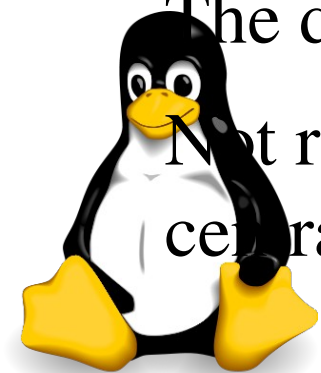
Can commit willy nilly on your own repo.

“Merge” or “rebase” or “pull” or “fetch” or “cherry pick” (you get the point) from other repos.

COMPLICATED. ONLY KERNEL HACKERS COULD UNDERSTAND!

The dudes with long hair in the basements of Siberia.

Not really that bad, but takes more effort to learn than centralized. Also more powerful.

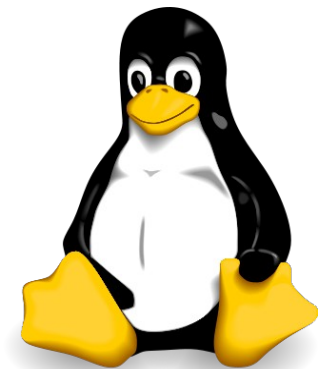


# SVN Symantics

---

```
# svnadmin create /repos/NEWREPO
```

Creates a new repository at /repos/NEWREPO



# Checking out (not in the lewd sense)

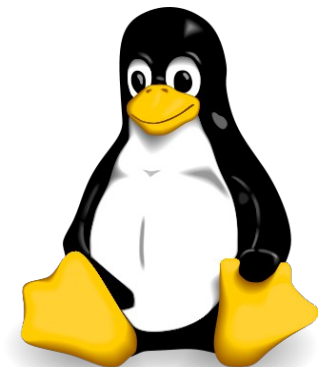
---

```
/projects # svn co <path-to-repo>
```

Copys all of the data in the repo to path.

Includes most recent revision of code and all the diffs.

Diffs and other data stored in “.svn” folder

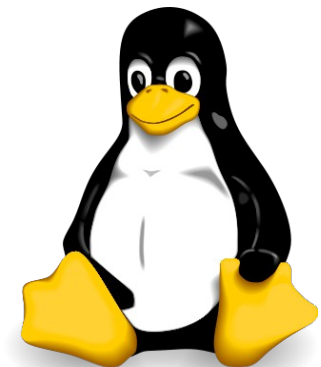


# Committing

```
/projects/stuff # svn commit (svn ci)
```

Commits to the repository the difference between the files as they are now and how they were at the most recent revision

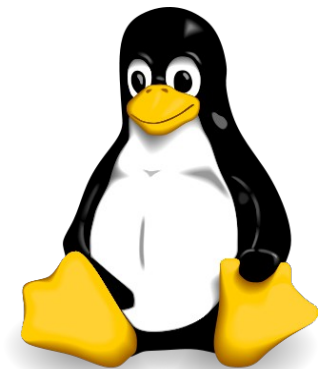
Others can then update to this new revision and get your changes



# Updating

```
/projects/stuff # svn update (-r ##)
```

Updates the current files to the most recent (or ##) revision.



# Other

---

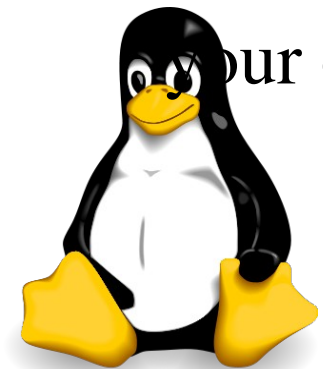
man svn

Or Google for other information.

Beware: There are *lots* of opinions on version control systems out there.

Ignore all of them.

Read the projects respective websites (svn, git etc.) and draw your own conclusions as to which to use for what size project.



# Rounding off

Fun with CDs!

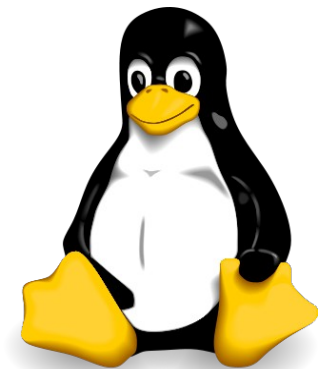
```
eject -T /dev/cdrom0
```

```
eject -T /dev/cdrom0
```

```
eject -T /dev/cdrom0
```

...

Guess what this does....

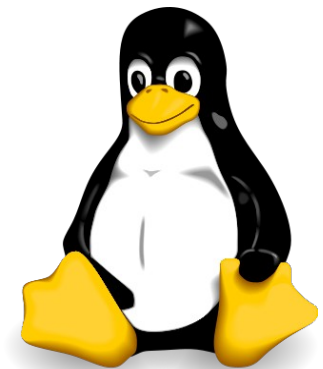


# LSOF

---

“lsof” ls open files

Useful for finding out why you can't unmount a file system, or who is using what data on a multiuser system.



# Netstat / nmap / nc

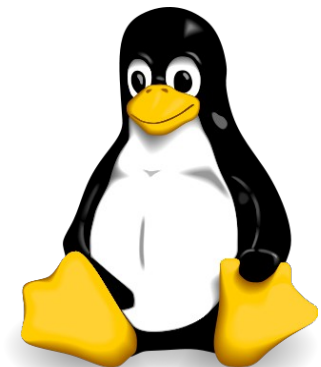
---

## Network toolchest

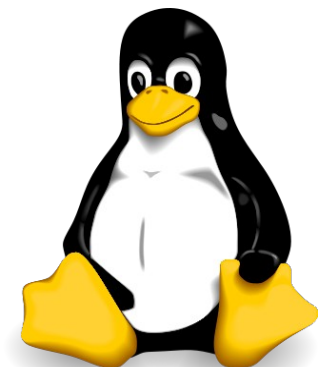
Netstat – list open sockets

Nmap – probe for open ports on a machine

Raw in/out on any given port



```
1 /tcp      open      hosts2-ns
0 [mobile]
1 Starting nmap U. 2.54BETA25
1 Insufficient responses for TCP sequencing (3), OS detection may be less
3 accurate
3 Interesting ports on 10.2.2.2:
3 (The 1539 ports scanned but not shown below are in state: closed)
4 Port      State      Service
4 22/tcp    open      ssh
1
1 No exact OS matches for host
8
8 Nmap run completed -- 1 IP address (1 host up) scanned
8 # sshnuke 10.2.2.2 -rootpw="210N0101"
4 Connecting to 10.2.2.2:ssh ... successful.
0 Attempting to exploit SSHv1 CRC32 ... successful.
  Reseting root password to "210N0101".
e System open: Access Level <9>
P # ssh 10.2.2.2 -l root
  root@10.2.2.2's password:
7
PRE-CONTROL> disable grid nodes 21 - 48
```



# Program debugging toolchest

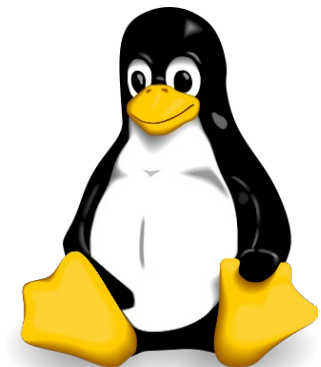
---

System trace – strace

Prints every time your app makes a kernel call

Library Trace – ltrace

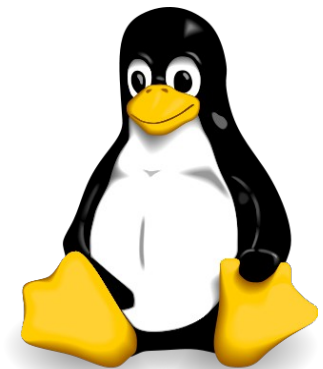
Prints every time your app calls into a library



# A brief aside...

---

[http://www.youtube.com/watch?v=ojFFS\\_T3UQk](http://www.youtube.com/watch?v=ojFFS_T3UQk)



# Perl Modules

<http://search.cpan.org>

**LOTS of Modules to do ‘stuff’**

**More than 12,200 Modules**

**Including LWP, WWW::Mechanize**