Packages

ComS 252 — Iowa State University

Andrew Miner

Overview

- You need to manage software packages on your system
  - Install new packages
  - Update/upgrade packages
  - Remove unused packages
  - All of this done as `root`

- This lecture covers three ways to do this
  1. High-level package managers (easiest)
  2. Low-level package managers
  3. Building from source code (hardest)
     Most of lecture concerns this way

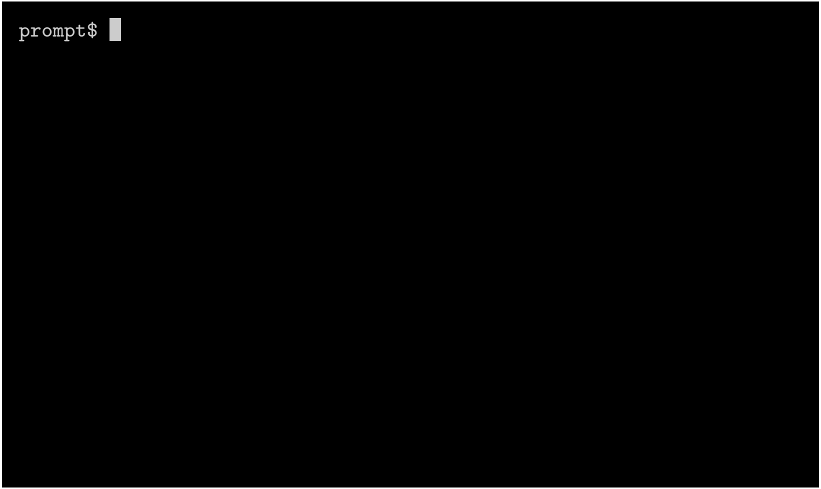
High-level package managers

- Work with software repositories (collections of packages)
  - Usually obtained over the network or Internet
- Can automatically fetch latest version of package(s)
- Can automatically install any dependencies
- Can automatically upgrade some or all packages
- Examples:

apt	aptitude
	For Debian-based systems
yum	Yellowdog updater, modified
	For Red hat systems
dnf	Dandified Yum
	Backward compatible replacement for `yum`

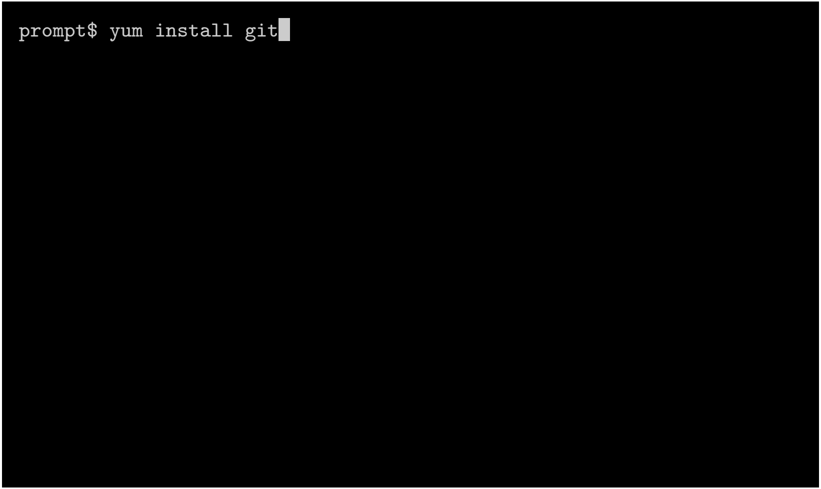
- There are various GUI front-ends for these

Example: using yum



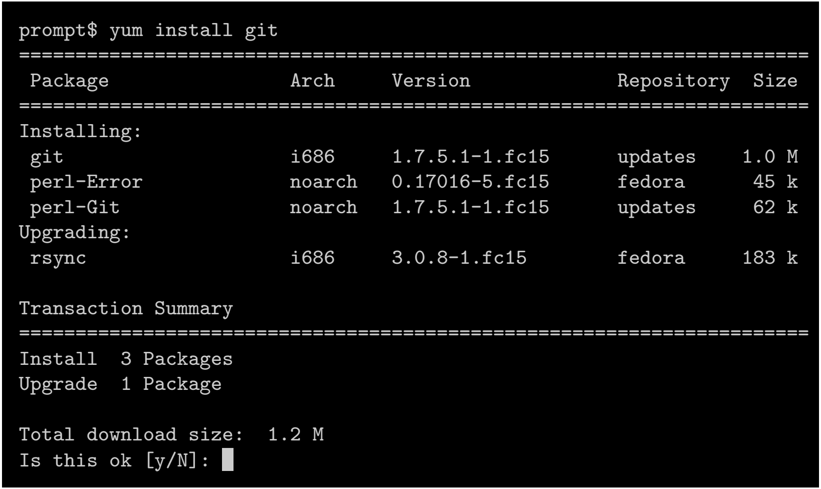
```
prompt$ 
```

Example: using yum



```
prompt$ yum install git
```

Example: using yum

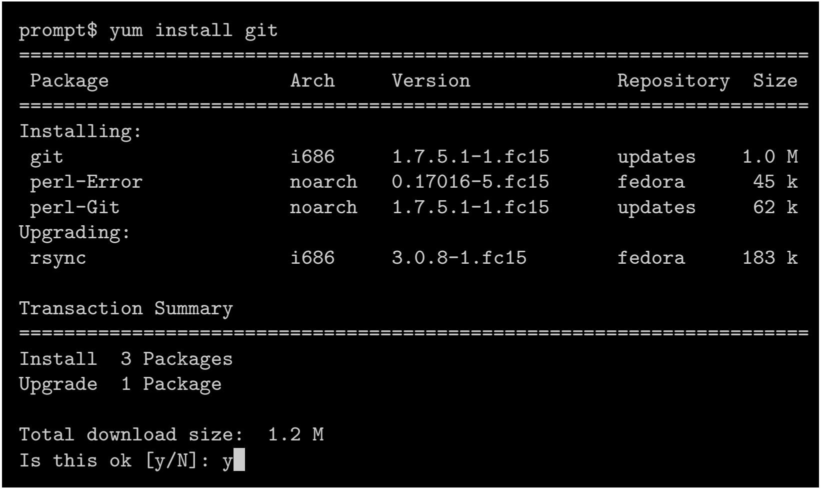


```
prompt$ yum install git
======================================================================
 Package                Arch     Version              Repository  Size
======================================================================
Installing:
 git                    i686     1.7.5.1-1.fc15       updates    1.0 M
 perl-Error             noarch   0.17016-5.fc15       fedora      45 k
 perl-Git               noarch   1.7.5.1-1.fc15       updates     62 k
Upgrading:
 rsync                  i686     3.0.8-1.fc15         fedora     183 k

Transaction Summary
======================================================================
Install   3 Packages
Upgrade   1 Package

Total download size:  1.2 M
Is this ok [y/N]:
```

Example: using yum

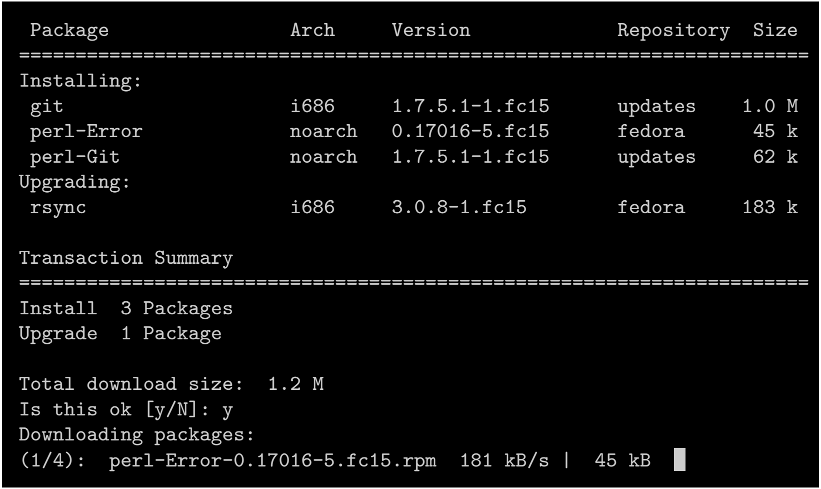


```
prompt$ yum install git
======================================================================
 Package                Arch     Version                 Repository  Size
======================================================================
Installing:
 git                    i686     1.7.5.1-1.fc15      updates    1.0 M
 perl-Error             noarch   0.17016-5.fc15      fedora      45 k
 perl-Git               noarch   1.7.5.1-1.fc15      updates     62 k
Upgrading:
 rsync                  i686     3.0.8-1.fc15        fedora     183 k

Transaction Summary
======================================================================
Install  3 Packages
Upgrade  1 Package

Total download size:  1.2 M
Is this ok [y/N]: y
```

Example: using yum

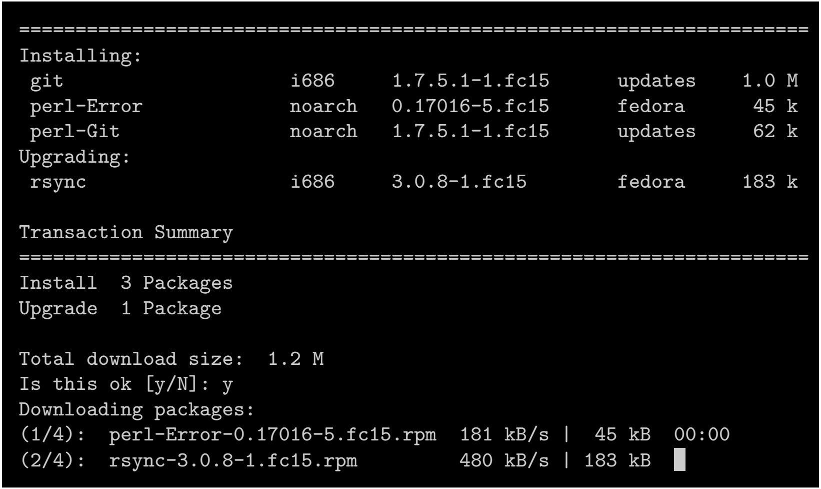


```
 Package                Arch     Version              Repository  Size
======================================================================
Installing:
 git                    i686     1.7.5.1-1.fc15       updates     1.0 M
 perl-Error             noarch   0.17016-5.fc15       fedora       45 k
 perl-Git               noarch   1.7.5.1-1.fc15       updates      62 k
Upgrading:
 rsync                  i686     3.0.8-1.fc15         fedora      183 k

Transaction Summary
======================================================================
Install   3 Packages
Upgrade   1 Package

Total download size:  1.2 M
Is this ok [y/N]: y
Downloading packages:
(1/4):  perl-Error-0.17016-5.fc15.rpm  181 kB/s |  45 kB  █
```

Example: using yum

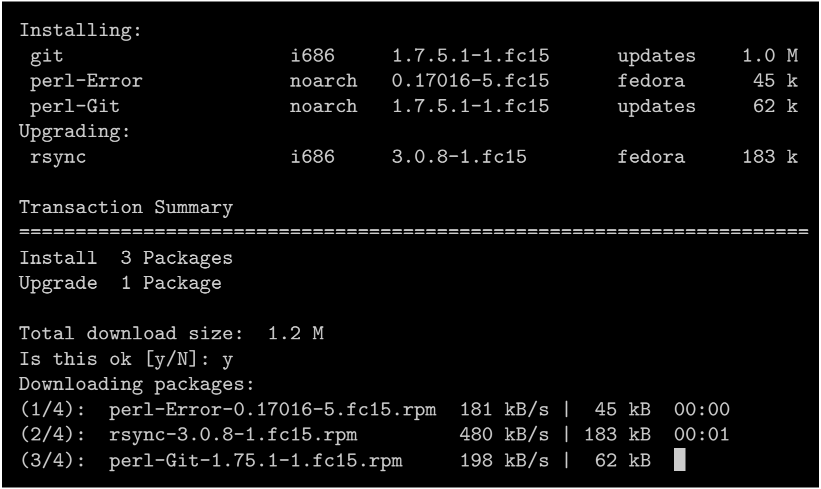


```
======================================================================
Installing:
 git                    i686     1.7.5.1-1.fc15      updates    1.0 M
 perl-Error             noarch   0.17016-5.fc15      fedora      45 k
 perl-Git               noarch   1.7.5.1-1.fc15      updates     62 k
Upgrading:
 rsync                  i686     3.0.8-1.fc15        fedora     183 k

Transaction Summary
======================================================================
Install   3 Packages
Upgrade   1 Package

Total download size:  1.2 M
Is this ok [y/N]: y
Downloading packages:
(1/4):  perl-Error-0.17016-5.fc15.rpm  181 kB/s |  45 kB  00:00
(2/4):  rsync-3.0.8-1.fc15.rpm         480 kB/s | 183 kB
```

Example: using yum

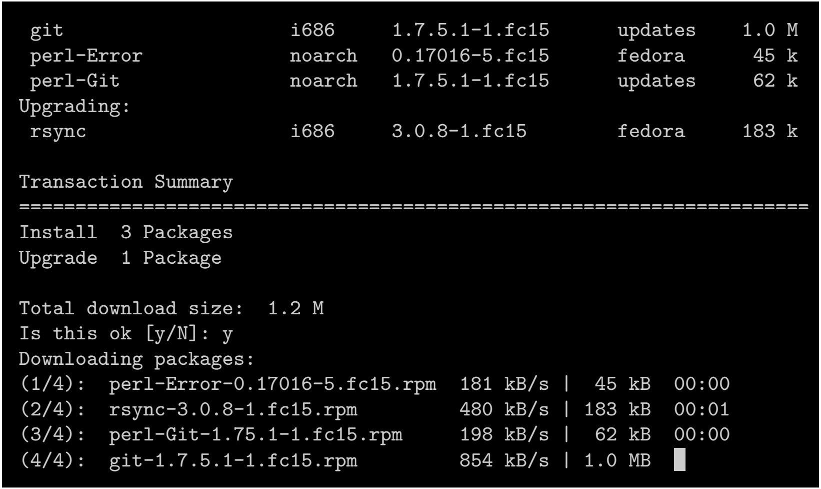


```
Installing:
 git                    i686     1.7.5.1-1.fc15      updates    1.0 M
 perl-Error             noarch   0.17016-5.fc15      fedora      45 k
 perl-Git               noarch   1.7.5.1-1.fc15      updates     62 k
Upgrading:
 rsync                  i686     3.0.8-1.fc15        fedora     183 k

Transaction Summary
======================================================================
Install   3 Packages
Upgrade   1 Package

Total download size:  1.2 M
Is this ok [y/N]: y
Downloading packages:
(1/4):  perl-Error-0.17016-5.fc15.rpm  181 kB/s |  45 kB  00:00
(2/4):  rsync-3.0.8-1.fc15.rpm         480 kB/s | 183 kB  00:01
(3/4):  perl-Git-1.75.1-1.fc15.rpm     198 kB/s |  62 kB  █
```

Example: using yum

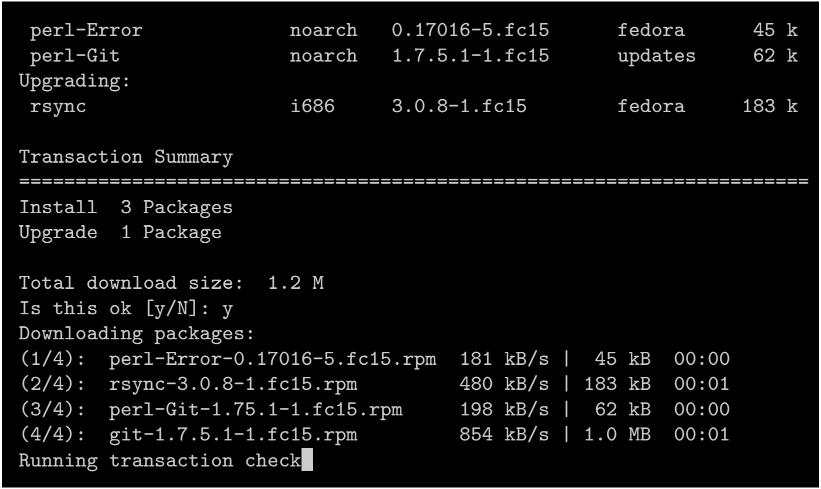


```
 git                    i686     1.7.5.1-1.fc15      updates    1.0 M
 perl-Error             noarch   0.17016-5.fc15      fedora      45 k
 perl-Git               noarch   1.7.5.1-1.fc15      updates     62 k
Upgrading:
 rsync                  i686     3.0.8-1.fc15        fedora     183 k

Transaction Summary
======================================================================
Install   3 Packages
Upgrade   1 Package

Total download size:  1.2 M
Is this ok [y/N]: y
Downloading packages:
(1/4):  perl-Error-0.17016-5.fc15.rpm  181 kB/s |  45 kB  00:00
(2/4):  rsync-3.0.8-1.fc15.rpm         480 kB/s | 183 kB  00:01
(3/4):  perl-Git-1.75.1-1.fc15.rpm     198 kB/s |  62 kB  00:00
(4/4):  git-1.7.5.1-1.fc15.rpm         854 kB/s | 1.0 MB
```

Example: using yum

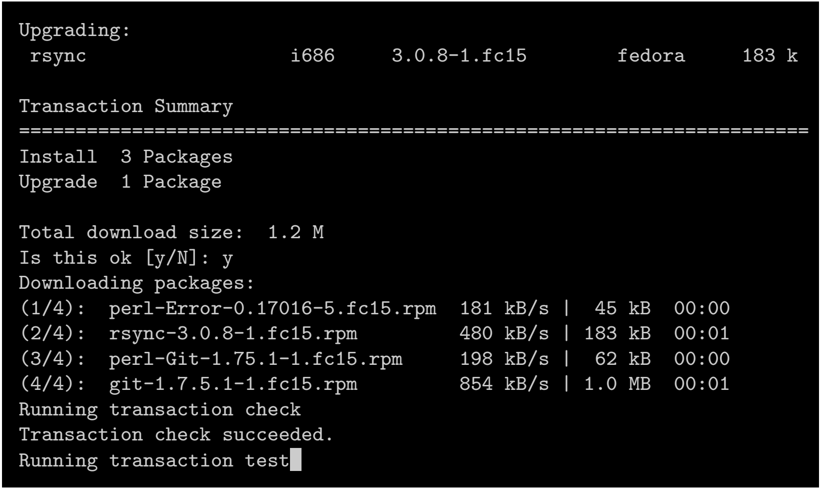


```
 perl-Error             noarch   0.17016-5.fc15      fedora      45 k
 perl-Git               noarch   1.7.5.1-1.fc15      updates     62 k
Upgrading:
 rsync                  i686     3.0.8-1.fc15        fedora     183 k

Transaction Summary
======================================================================
Install  3 Packages
Upgrade  1 Package

Total download size:  1.2 M
Is this ok [y/N]: y
Downloading packages:
(1/4):  perl-Error-0.17016-5.fc15.rpm  181 kB/s |  45 kB  00:00
(2/4):  rsync-3.0.8-1.fc15.rpm         480 kB/s | 183 kB  00:01
(3/4):  perl-Git-1.75.1-1.fc15.rpm     198 kB/s |  62 kB  00:00
(4/4):  git-1.7.5.1-1.fc15.rpm         854 kB/s | 1.0 MB  00:01
Running transaction check
```

Example: using yum

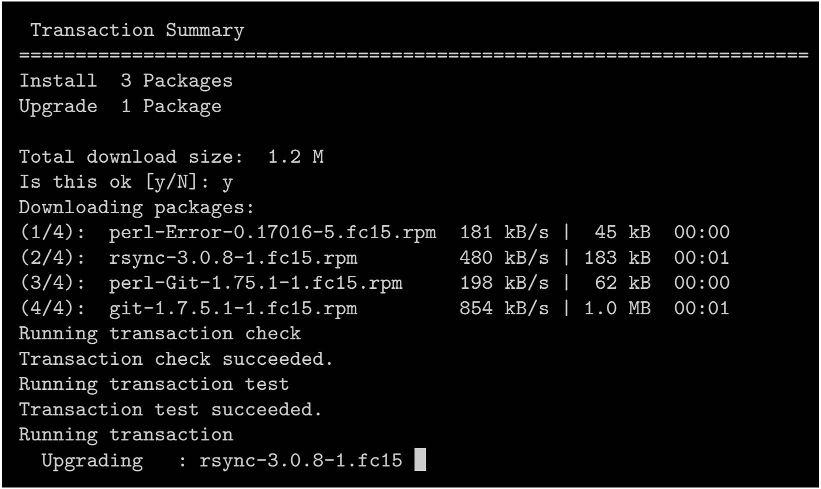


```
Upgrading:
 rsync                  i686     3.0.8-1.fc15        fedora     183 k

Transaction Summary
======================================================================
Install  3 Packages
Upgrade  1 Package

Total download size:  1.2 M
Is this ok [y/N]: y
Downloading packages:
(1/4):  perl-Error-0.17016-5.fc15.rpm  181 kB/s |  45 kB  00:00
(2/4):  rsync-3.0.8-1.fc15.rpm         480 kB/s | 183 kB  00:01
(3/4):  perl-Git-1.75.1-1.fc15.rpm     198 kB/s |  62 kB  00:00
(4/4):  git-1.7.5.1-1.fc15.rpm         854 kB/s | 1.0 MB  00:01
Running transaction check
Transaction check succeeded.
Running transaction test
```

Example: using yum



```
 Transaction Summary
======================================================================
Install   3 Packages
Upgrade   1 Package

Total download size:  1.2 M
Is this ok [y/N]: y
Downloading packages:
(1/4):  perl-Error-0.17016-5.fc15.rpm  181 kB/s |  45 kB  00:00
(2/4):  rsync-3.0.8-1.fc15.rpm         480 kB/s | 183 kB  00:01
(3/4):  perl-Git-1.75.1-1.fc15.rpm     198 kB/s |  62 kB  00:00
(4/4):  git-1.7.5.1-1.fc15.rpm         854 kB/s | 1.0 MB  00:01
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
  Upgrading    : rsync-3.0.8-1.fc15 
```

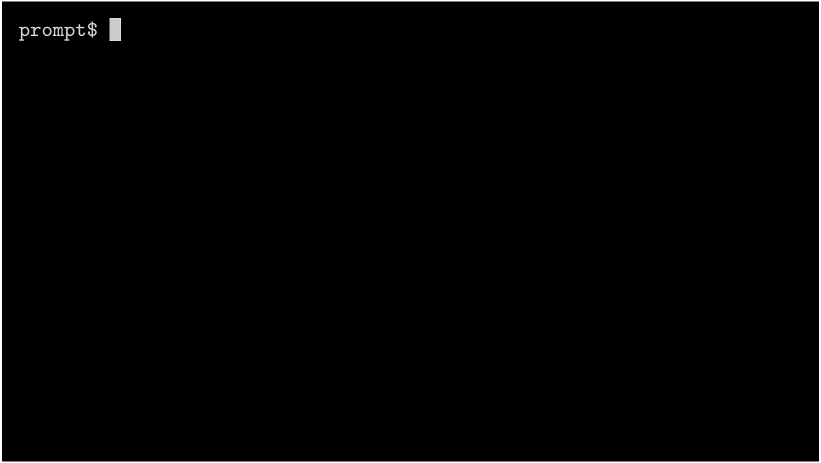
Low-level package managers

- Work with package files
- Package files track dependencies
  - Tells you which packages must be installed first
- Do not automatically fetch dependencies
- . . . but, you can install several package files at once
- Examples:

dpkg	Debian Package Handles `.deb` packages
rpm	Red Hat Package Manager Handles `.rpm` packages

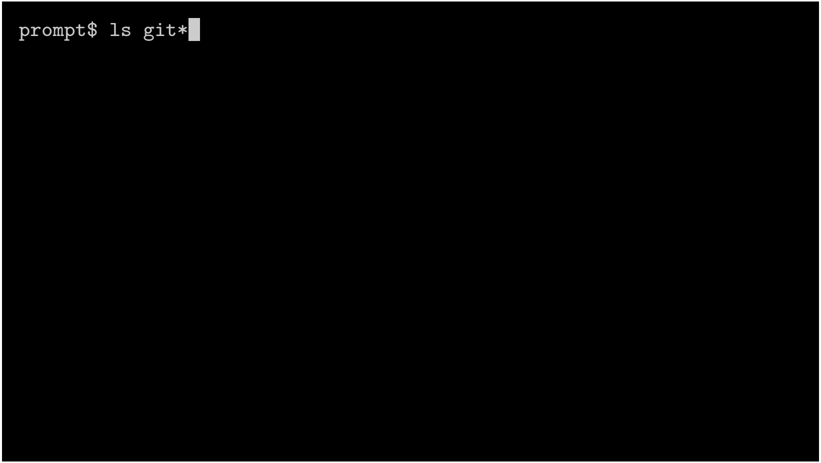
- High-level package managers are just nice front-ends for these

Example: using rpm



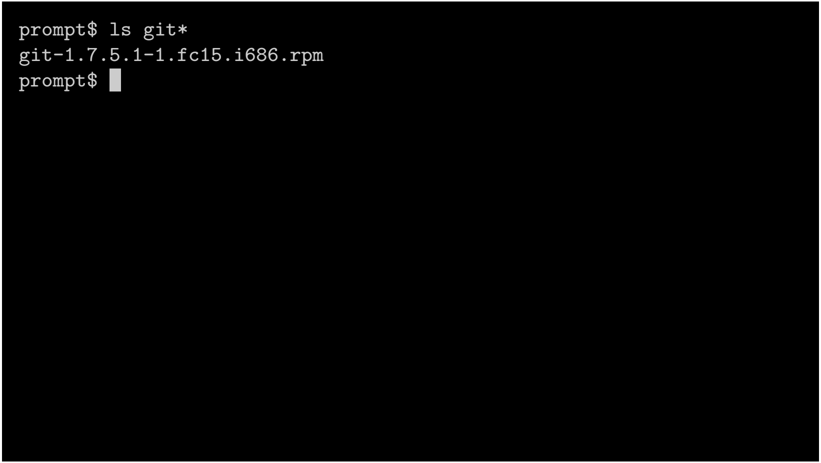
```
prompt$ █
```

Example: using rpm



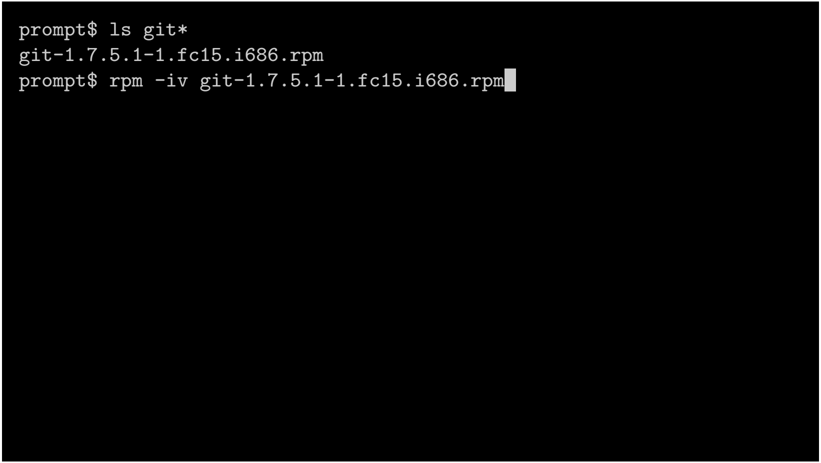
```
prompt$ ls git*
```

Example: using rpm



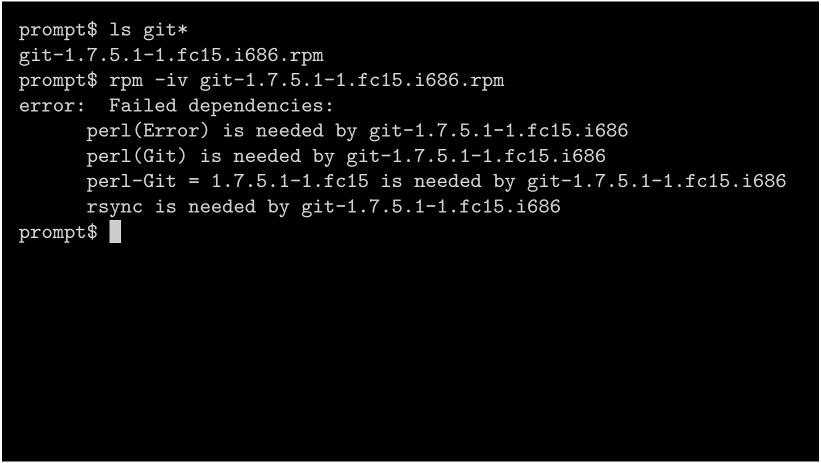
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ █
```

Example: using rpm



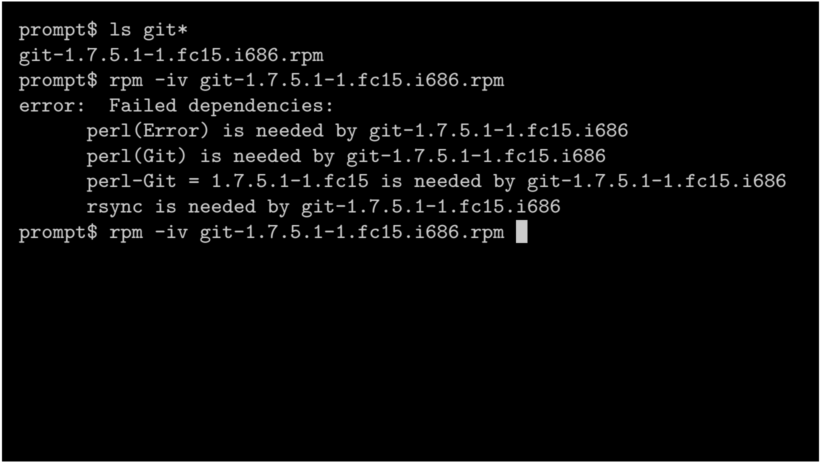
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
```

Example: using rpm



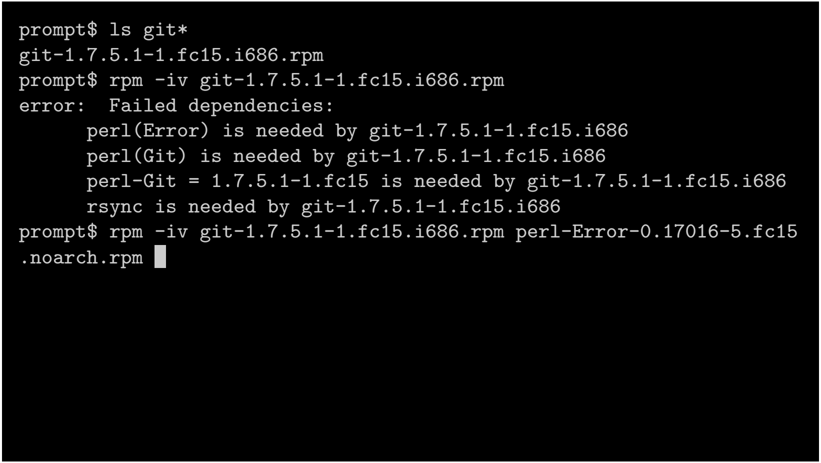
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ █
```

Example: using rpm



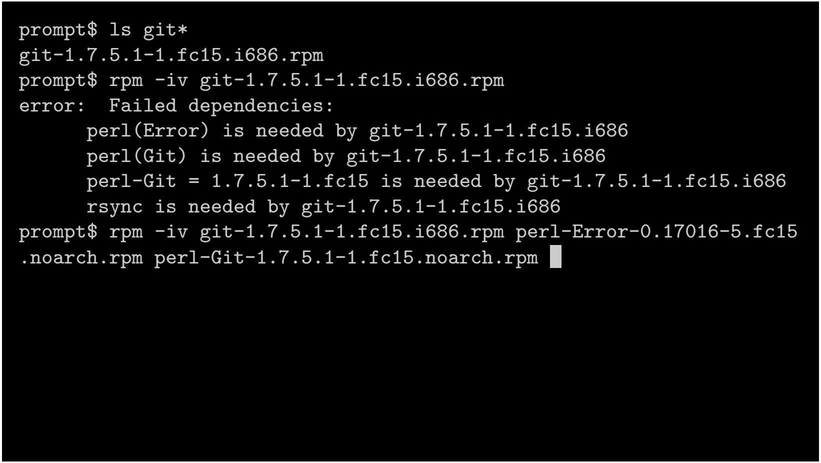
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
```

Example: using rpm



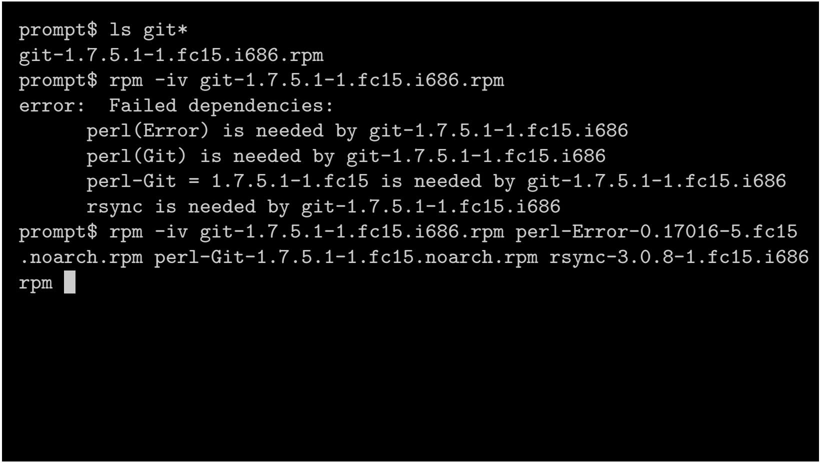
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm perl-Error-0.17016-5.fc15
.noarch.rpm
```

Example: using rpm



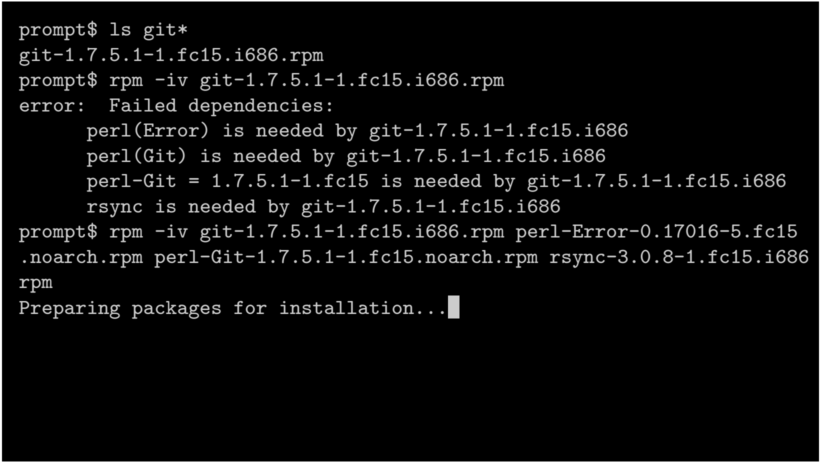
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm perl-Error-0.17016-5.fc15
.noarch.rpm perl-Git-1.7.5.1-1.fc15.noarch.rpm
```

Example: using rpm



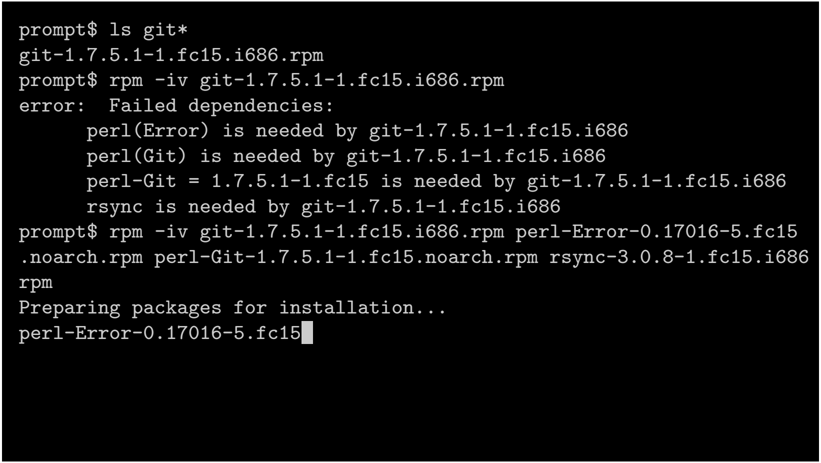
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm perl-Error-0.17016-5.fc15
.noarch.rpm perl-Git-1.7.5.1-1.fc15.noarch.rpm rsync-3.0.8-1.fc15.i686
rpm
```

Example: using rpm



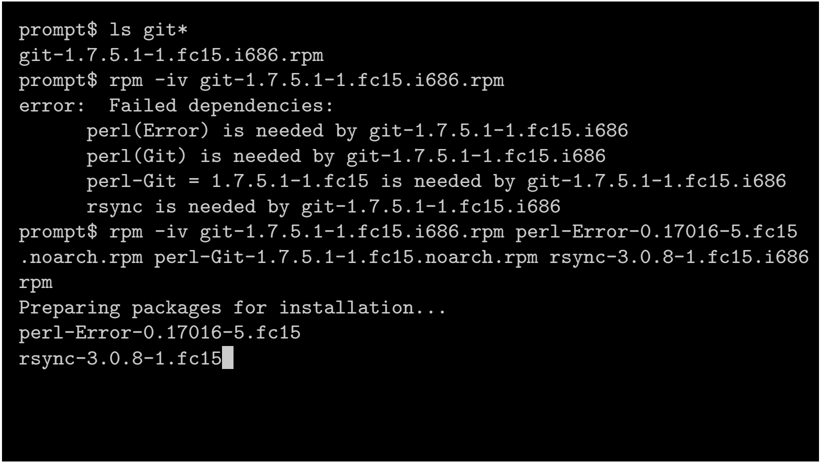
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm perl-Error-0.17016-5.fc15
.noarch.rpm perl-Git-1.7.5.1-1.fc15.noarch.rpm rsync-3.0.8-1.fc15.i686
rpm
Preparing packages for installation...
```

Example: using rpm



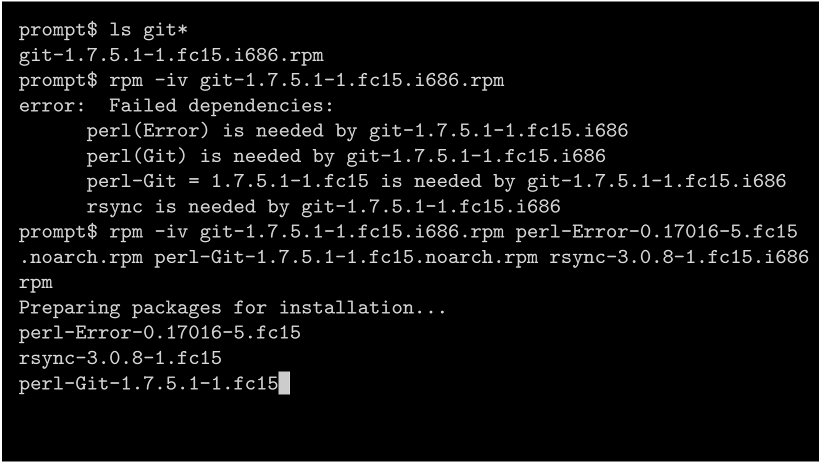
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm perl-Error-0.17016-5.fc15
.noarch.rpm perl-Git-1.7.5.1-1.fc15.noarch.rpm rsync-3.0.8-1.fc15.i686
rpm
Preparing packages for installation...
perl-Error-0.17016-5.fc15
```

Example: using rpm



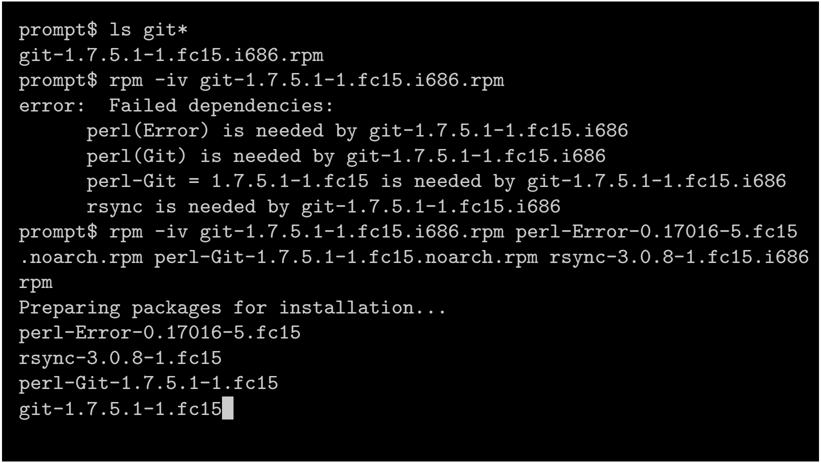
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm perl-Error-0.17016-5.fc15
.noarch.rpm perl-Git-1.7.5.1-1.fc15.noarch.rpm rsync-3.0.8-1.fc15.i686
rpm
Preparing packages for installation...
perl-Error-0.17016-5.fc15
rsync-3.0.8-1.fc15
```

Example: using rpm



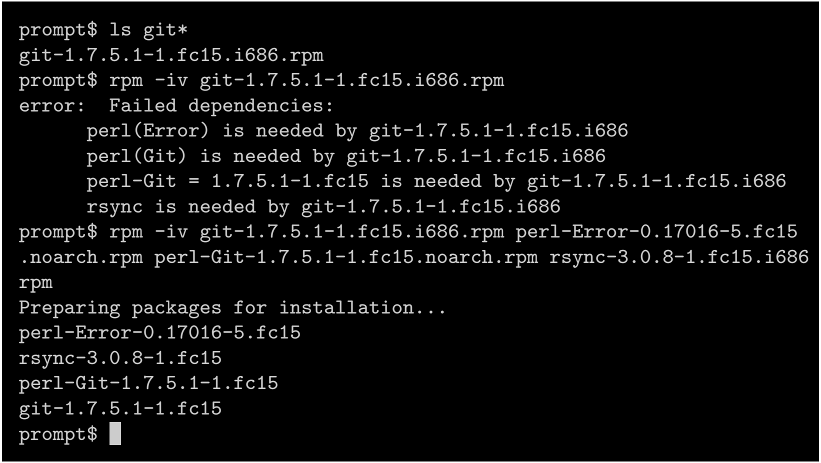
```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm perl-Error-0.17016-5.fc15
.noarch.rpm perl-Git-1.7.5.1-1.fc15.noarch.rpm rsync-3.0.8-1.fc15.i686
rpm
Preparing packages for installation...
perl-Error-0.17016-5.fc15
rsync-3.0.8-1.fc15
perl-Git-1.7.5.1-1.fc15
```

Example: using `rpm`



```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm perl-Error-0.17016-5.fc15
.noarch.rpm perl-Git-1.7.5.1-1.fc15.noarch.rpm rsync-3.0.8-1.fc15.i686
rpm
Preparing packages for installation...
perl-Error-0.17016-5.fc15
rsync-3.0.8-1.fc15
perl-Git-1.7.5.1-1.fc15
git-1.7.5.1-1.fc15
```

Example: using rpm



```
prompt$ ls git*
git-1.7.5.1-1.fc15.i686.rpm
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm
error:  Failed dependencies:
      perl(Error) is needed by git-1.7.5.1-1.fc15.i686
      perl(Git) is needed by git-1.7.5.1-1.fc15.i686
      perl-Git = 1.7.5.1-1.fc15 is needed by git-1.7.5.1-1.fc15.i686
      rsync is needed by git-1.7.5.1-1.fc15.i686
prompt$ rpm -iv git-1.7.5.1-1.fc15.i686.rpm perl-Error-0.17016-5.fc15
.noarch.rpm perl-Git-1.7.5.1-1.fc15.noarch.rpm rsync-3.0.8-1.fc15.i686
rpm
Preparing packages for installation...
perl-Error-0.17016-5.fc15
rsync-3.0.8-1.fc15
perl-Git-1.7.5.1-1.fc15
git-1.7.5.1-1.fc15
prompt$ 
```

Why build from source code?

- Because you can
- You are porting from another system (Unix or even Windows)
- There is a problem with the package or binary
- There is no package file or binary
- You want a newer version than the package file
- You want a custom configuration of the package
- You want to modify the source code
- Required for homework

Why not build from source code?

- `yum install package` is way easier
- You have to track dependencies by hand
- You need to build the software[1], which means either
  1. You build the software on the target machine
     - The system must have development tools installed. Usually, `gcc` and `make`, possibly others
     - The system needs to be powerful enough to do the build
  2. You build the software somewhere else, and copy it over
     - This might mean a cross compiler
     - . . . unless they are similar systems
- You get to deal with configuration and compile errors yourself
  - More on this later

[1] Most system software is written in C.

Generic steps to build from source code

1. Obtain source code
2. Read documentation
    - Look for text files named `README`, `INSTALL`
    - `COPYING`: license information
3. Configure
4. Build
5. Test
    - Just because it builds, doesn't mean it works
6. Install
7. Enjoy

Generic steps to build from source code

1. Obtain source code
2. Read documentation
   - Look for text files named `README`, `INSTALL`
   - `COPYING`: license information
3. Configure
4. Build
5. Test
   - Just because it builds, doesn't mean it works
6. Install
7. Enjoy

Let's talk about these

Finding source code

- I will assume that it is available over the Internet
  - Sourceforge, for example
- So, how hard is it to download over the Internet?
  - Typical projects are multiple source files
    - Can be hundreds
  - Not fun to download these one at a time, by hand
- Ok, so we need a nice way to get lots of source files

tar: tape archive

- Utility to manage archives
- Can preserve directories, links, owners, groups, permissions
- Usage is a little tricky.
- Mandatory main switches (must specify exactly one):

  `c` : create new archive
  `r` : append to existing archive
  `t` : list contents
  `x` : extract files
  ... : (there are others)
- An extremely useful optional switch (check your `man` pages):

  `f` : specify a file instead of the tape drive
  - use "–" for standard input/output
  - Tar archive files are called tarballs

Simple tar examples

- Create an archive stored in file `foo.tar`,

  that contains a copy of directories `Project1`, `Project2`

```
tar cf foo.tar Project1 Project2
```

- List contents of archive `foo.tar`

```
tar tf foo.tar
```

- Extract (copies of) files from archive `foo.tar`

```
tar xf foo.tar
```

  The archive is not modified when you do this

More useful `tar` switches

`v` : verbose

- Show more information
  (e.g., long listing when using `t`)

`z` : compress / uncompress using `gzip`/`gunzip`

`j` : compress / uncompress using `bzip2`/`bunzip2`

The following commands produce *identical* compressed tarballs:

1.

```
tar cf file.tar SourceDir && gzip file.tar
```

2.

```
tar cf - SourceDir | gzip > file.tar.gz
```

3.

```
tar czf file.tar.gz SourceDir
```

Summary so far

Ok, so how do we obtain source code using `tar`?

Summary so far

Ok, so how do we obtain source code using `tar`?

1. Download a *single* file (a compressed tarball)

   `coolthing-2.0.4.tar.gz`

Summary so far

Ok, so how do we obtain source code using `tar`?

1. Download a *single* file (a compressed tarball)
   `coolthing-2.0.4.tar.gz`
2. Unpack the tarball

Summary so far

Ok, so how do we obtain source code using `tar`?

1. Download a *single* file (a compressed tarball)
   `coolthing-2.0.4.tar.gz`
2. Unpack the tarball

```
tar xf coolthing-2.0.4.tar.gz
```

Summary so far

Ok, so how do we obtain source code using `tar`?

1. Download a *single* file (a compressed tarball)

   `coolthing-2.0.4.tar.gz`
2. Unpack the tarball

```
tar xf coolthing-2.0.4.tar.gz
```

If you have an older version of `tar`, must use:

```
tar xzf coolthing-2.0.4.tar.gz
```

Summary so far

Ok, so how do we obtain source code using tar?

1. Download a *single* file (a compressed tarball)

   coolthing-2.0.4.tar.gz
2. Unpack the tarball

```
tar xf coolthing-2.0.4.tar.gz
```

If you have an older version of tar, must use:

```
tar xzf coolthing-2.0.4.tar.gz
```

Are there other ways to obtain source code?

Summary so far

Ok, so how do we obtain source code using `tar`?

1. Download a *single* file (a compressed tarball)

   `coolthing-2.0.4.tar.gz`
2. Unpack the tarball

```
tar xf coolthing-2.0.4.tar.gz
```

If you have an older version of `tar`, must use:

```
tar xzf coolthing-2.0.4.tar.gz
```

Are there other ways to obtain source code?

- Yes: other utilities like `tar`
  - `zip` and `unzip` (less common for source code)
  - There are “source RPMs”

Summary so far

Ok, so how do we obtain source code using `tar`?

1. Download a *single* file (a compressed tarball)

   `coolthing-2.0.4.tar.gz`
2. Unpack the tarball

```
tar xf coolthing-2.0.4.tar.gz
```

If you have an older version of `tar`, must use:

```
tar xzf coolthing-2.0.4.tar.gz
```

Are there other ways to obtain source code?

- Yes: other utilities like `tar`
  - `zip` and `unzip` (less common for source code)
  - There are "source RPMs"
- Yes: other mechanisms

Versioning software

- *Essential* for online development
- Source code and all revisions are is stored in a repository
  - Done in a clever way — just the changes between versions
- Developers work on their own working copy
  - Can easily update working copy to latest or any version
    - Only sends differences
  - Can easily upload working copy changes into the repository
    - Only sends differences
- Conflicts can be detected
- Systems for this include

RCS	:	Revision Control System (single files only)
CVS	:	Concurrent Versions System, replaces RCS
subversion	:	Replacement for CVS
git	:	Alternate system designed by Linus Torvalds

How is “versioning software” relevant?

You can often obtain the source code directly from the repository

- You can get the *latest* version of the code
- Pretty easy to do, e.g.

```
svn checkout url Working/Copy/Dir
```

- Easy to update to newer versions, later

However:

- You won’t be able to upload any changes
- Requires Internet connection

For more info:

- `http://svnbook.red-bean.com/`
- `http://git-scm.com/`

Some related utilities

diff: compare files line by line

- Lots of switches — check your man pages
  - Can compare directories, recursively
- Writes file differences to standard output
- Output is a bit cryptic. . .

```
prompt$ █
```

Some related utilities

diff: compare files line by line

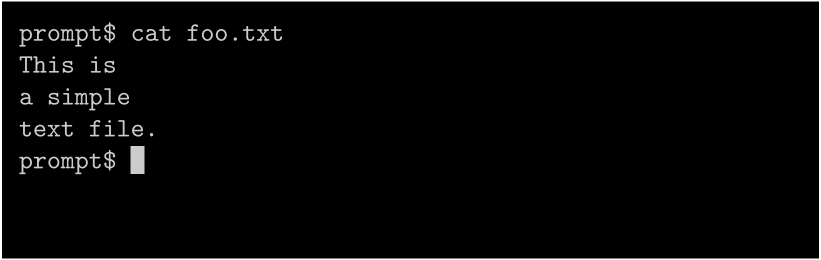
- Lots of switches — check your man pages
    - Can compare directories, recursively
- Writes file differences to standard output
- Output is a bit cryptic. . .

```
prompt$ cat foo.txt
```

Some related utilities

diff: compare files line by line

- Lots of switches — check your man pages
  - Can compare directories, recursively
- Writes file differences to standard output
- Output is a bit cryptic. . .

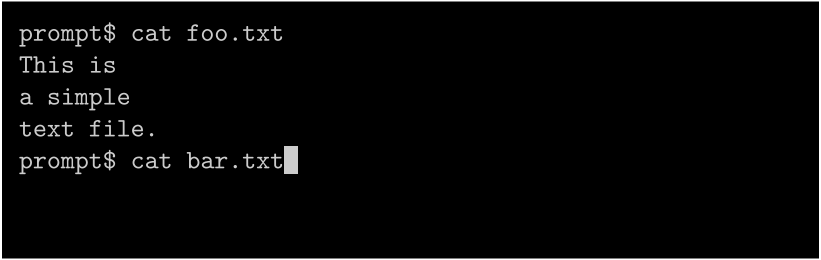


```
prompt$ cat foo.txt
This is
a simple
text file.
prompt$ █
```

Some related utilities

diff: compare files line by line

- Lots of switches — check your man pages
  - Can compare directories, recursively
- Writes file differences to standard output
- Output is a bit cryptic. . .

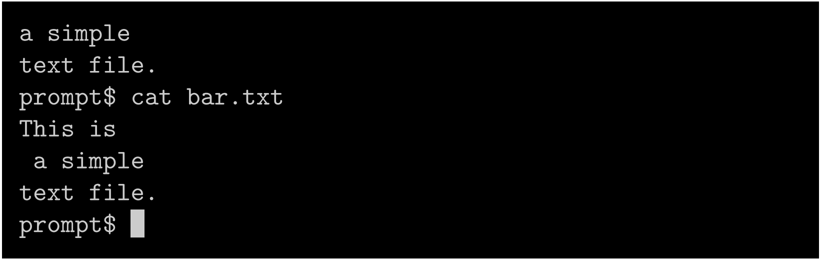


```
prompt$ cat foo.txt
This is
a simple
text file.
prompt$ cat bar.txt
```

Some related utilities

diff: compare files line by line

- Lots of switches — check your man pages
  - Can compare directories, recursively
- Writes file differences to standard output
- Output is a bit cryptic. . .



```
a simple
text file.
prompt$ cat bar.txt
This is
 a simple
text file.
prompt$ 
```

Some related utilities

diff: compare files line by line

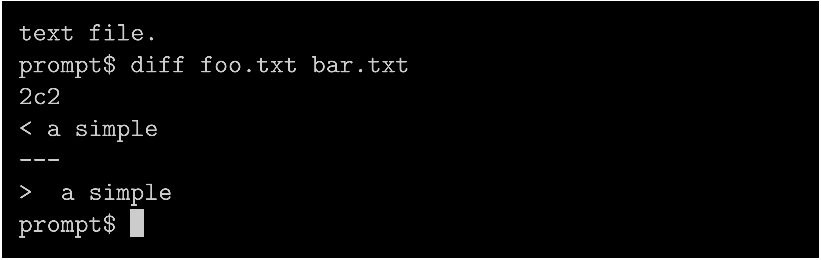
- Lots of switches — check your man pages
  - Can compare directories, recursively
- Writes file differences to standard output
- Output is a bit cryptic. . .

```
a simple
text file.
prompt$ cat bar.txt
This is
 a simple
text file.
prompt$ diff foo.txt bar.txt
```

Some related utilities

diff: compare files line by line

- Lots of switches — check your man pages
  - Can compare directories, recursively
- Writes file differences to standard output
- Output is a bit cryptic. . .

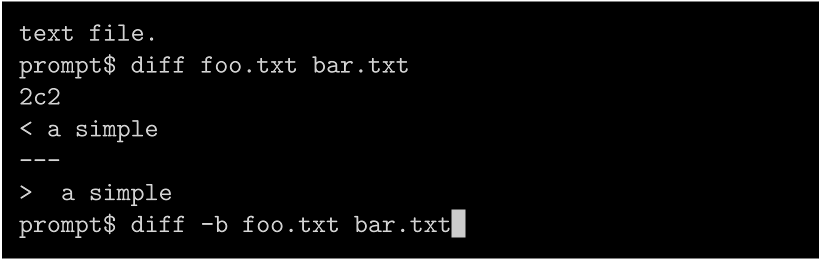


```
text file.
prompt$ diff foo.txt bar.txt
2c2
< a simple
---
>  a simple
prompt$ 
```

Some related utilities

diff: compare files line by line

- Lots of switches — check your man pages
  - Can compare directories, recursively
- Writes file differences to standard output
- Output is a bit cryptic. . .



```
text file.
prompt$ diff foo.txt bar.txt
2c2
< a simple
---
>  a simple
prompt$ diff -b foo.txt bar.txt
```

Some related utilities

`diff`: compare files line by line

- Lots of switches — check your `man` pages
    - Can compare directories, recursively
- Writes file differences to standard output
- Output is a bit cryptic...

```
prompt$ diff foo.txt bar.txt
2c2
< a simple
---
>  a simple
prompt$ diff -b foo.txt bar.txt
prompt$ █
```

Some related utilities

patch: apply a “patch” to a file

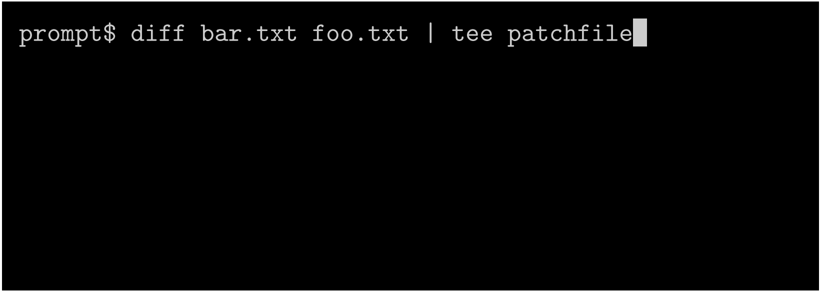
- Usage: `patch original changes`
- `changes` is a “patchfile”
  - Tells what changes to make, to original file
  - Output of `diff` is a patchfile

```
prompt$ █
```

Some related utilities

patch: apply a “patch” to a file

- Usage: `patch original changes`
- `changes` is a “patchfile”
  - Tells what changes to make, to original file
  - Output of `diff` is a patchfile

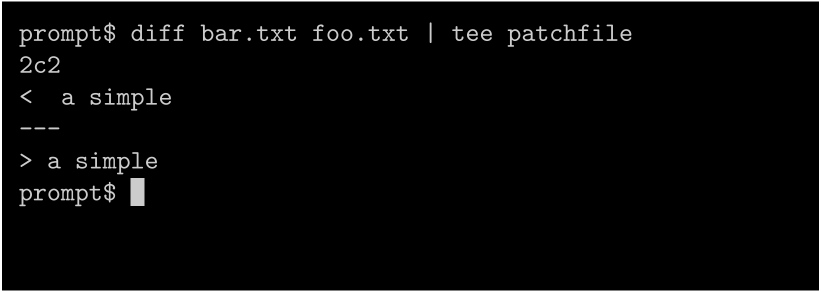


```
prompt$ diff bar.txt foo.txt | tee patchfile
```

Some related utilities

patch: apply a "patch" to a file

- Usage: `patch original changes`
- `changes` is a "patchfile"
  - Tells what changes to make, to original file
  - Output of `diff` is a patchfile

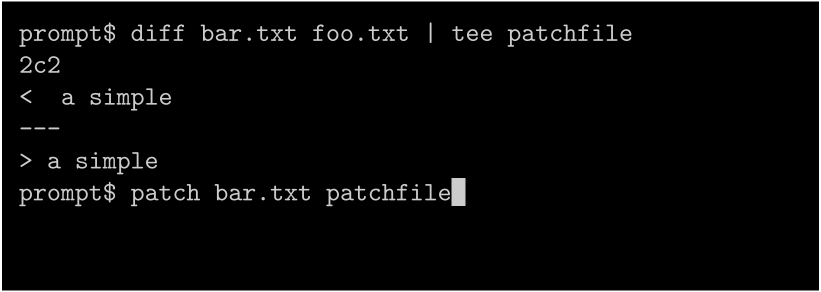


```
prompt$ diff bar.txt foo.txt | tee patchfile
2c2
<  a simple
---
> a simple
prompt$ █
```

Some related utilities

patch: apply a "patch" to a file

- Usage: `patch original changes`
- `changes` is a "patchfile"
  - Tells what changes to make, to original file
  - Output of `diff` is a patchfile



```
prompt$ diff bar.txt foo.txt | tee patchfile
2c2
<  a simple
---
> a simple
prompt$ patch bar.txt patchfile
```

Some related utilities

patch: apply a “patch” to a file

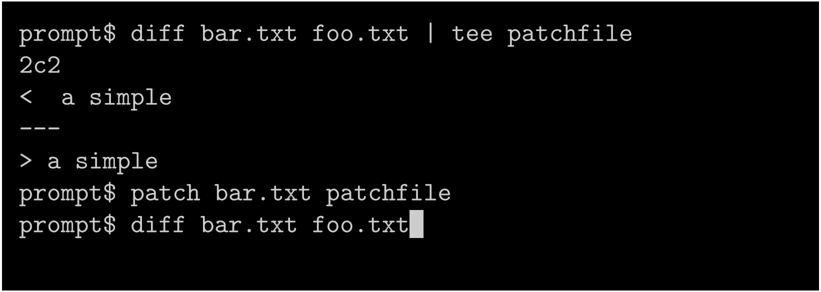
- Usage: `patch original changes`
- `changes` is a “patchfile”
  - Tells what changes to make, to original file
  - Output of `diff` is a patchfile

```
prompt$ diff bar.txt foo.txt | tee patchfile
2c2
<  a simple
---
> a simple
prompt$ patch bar.txt patchfile
prompt$ █
```

Some related utilities

patch: apply a “patch” to a file

- Usage: `patch original changes`
- `changes` is a “patchfile”
    - Tells what changes to make, to original file
    - Output of `diff` is a patchfile



```
prompt$ diff bar.txt foo.txt | tee patchfile
2c2
<  a simple
---
> a simple
prompt$ patch bar.txt patchfile
prompt$ diff bar.txt foo.txt
```

Some related utilities

patch: apply a “patch” to a file

- Usage: `patch original changes`
- `changes` is a “patchfile”
  - Tells what changes to make, to original file
  - Output of `diff` is a patchfile

```
prompt$ diff bar.txt foo.txt | tee patchfile
2c2
<  a simple
---
> a simple
prompt$ patch bar.txt patchfile
prompt$ diff bar.txt foo.txt
prompt$
```

Some related utilities

rsync: remote file copy

- Can "remotely synchronize" two directories
  - But you have to specify "which way" to copy
- Is smart — will compress and send differences between files
- Perfect for creating site "mirrors"

Some related utilities

rsync: remote file copy

- Can "remotely synchronize" two directories
  - But you have to specify "which way" to copy
- Is smart — will compress and send differences between files
- Perfect for creating site "mirrors"

So, what's the difference between `rsync` and, say, `subversion`?

Some related utilities

rsync: remote file copy

- Can "remotely synchronize" two directories
  - But you have to specify "which way" to copy
- Is smart — will compress and send differences between files
- Perfect for creating site "mirrors"

So, what's the difference between `rsync` and, say, `subversion`?

- `subversion` gives you version history
  - Can revert to previous versions
- `subversion` gives you conflict resolution
  - Ways to deal with two people changing the same file

Obtaining software, summary

Choose between

1. Download and unpack an archive file
   - Compressed tarball, source RPM, or other format
   - Might need to apply patches as well
2. Connect to a repository and check out a working copy
   - Using `CVS`, `subversion`, or `git`
3. Remote copy from somewhere using `rsync`

Not all of these choices may be available

- Depends on what developers make available

Configuring software — why do I need to?

- Supporting libraries may differ slightly on different systems
- Libraries may be in different locations
    - E.g., `/lib` or `/usr/lib` or `/usr/local/lib`?
- Which C standard is used in the source?
    - E.g., C90, C99, C11?
- There may be choices to enable or disable features
    - This is one reason to read the documentation

Common configuration models

None : the developer(s) provide nothing for configuration

- Easy for the developer
- Easy for you — unless it doesn't build

Static : choose from a few "canned" options

- For example,
  - On BSD, do this . . .
  - On Linux, do this . . .
  - On Darwin (Mac OS X), do this . . .
- Better than "none", anyway

Dynamic : run a program to determine system capabilities

- E.g., find location of required libraries
- Can often pass options to this program
- Did I mention — read the documentation?

Configuration programs

Common things you run for “dynamic” configuration

`xmkmf` : common for X applications

- Finds location of X headers, fonts, libraries, etc.

`./config` or

`./configure` : local configuration script

- Might be generated by another program you have to run first
- (Most) GNU software does this
  - Need “auto” tools: `automake`, `autoconf`, . . .
- Can pass options, some are “standard”, e.g.,

```
./configure CFLAGS=-O3 --without-gmp
```

- Read the documentation!

make

- Utility to update files
- Uses a set of rules, in a makefile
  - Usually, named `makefile` or `Makefile`
- Rules say how files can be built (targets)
- Rules specify dependencies
  - When dependencies change, target must be rebuilt

So, how do I use `make` to build software?

- Assuming the developers have done their jobs...
- Simply type "`make`"
- You should do this as an ordinary user, not `root`
  - Even for "system" software
- Read comics, check your mail, etc., while everything builds
- Read the documentation to be sure this is right

Typical useful `make` targets

make or

make all : should build everything

make clean : remove compiled binaries

make distclean : revert to “freshly unpacked tarball” state

- Useful if you need to start over, e.g., reconfigure and rebuild

make check : run tests (if developers made some)

make install : install

- You may want to do this by hand
- Or at least be sure it will install where you want
- May need to do this as root (depending on where it will be installed)

`make` and the configuration models

None : the developers give you a `makefile`

- Be wary if they do not

Static : typically, different `makefiles` for different systems

Dynamic : the `makefile` is built for you

- `xmkmf`
  - Produces a `makefile`
- `./configure`
  - Produces a `makefile`
  - Usually produces other things (e.g., `config.h`)

How to install

Choose one of the following.

1. `make install` (preferred)
2. Copy by hand

For “system software”, you will need to do this as `root`.

How to install

Choose one of the following.

1. `make install` (preferred)
2. Copy by hand

For “system software”, you will need to do this as `root`.

But, where should things go?

- Ordinary executables should go in some `bin`:

  `/bin`, `/usr/bin`, `/usr/local/bin`, or even `/opt/bin`
- “System management” executables should go in some `sbin`:

  `/sbin`, `/usr/sbin`, `/usr/local/sbin`
- Libraries should go in some `lib`:

  `/lib`, `/usr/lib`, `/usr/local/lib`, `/opt/lib`

Where to put things

http://en.wikipedia.org/wiki/Filesystem_Hierarchy_Standard

`/bin, /sbin, /lib`

- Essential utilities and libraries
- E.g., things *always* needed, like `bash`, `ls`, `cp`

`/usr/bin, /usr/sbin, /usr/lib`

- Utilities and libraries, maintained by distribution
- Things like `yum`, `firefox`

`/usr/local/bin, /usr/local/sbin, /usr/local/lib`

- Utilities and libraries, "local to this machine"
- Or things that you install yourself

`/opt/bin, /opt/lib, /opt/local/bin, /opt/local/lib`

- Things that don't integrate well
- Or things that you install yourself

`~/bin, ~/lib`

- For "personal" copies of software

Typical ./configure directory settings

http://www.gnu.org/prep/standards/html_node/Directory-Variables.html

prefix

- Prefix for where things should be installed
- E.g., /usr, /usr/local
- Does not include the final bin or lib

exec_prefix

- Often, the same as prefix
- Prefix for executables and libraries

Example:

```
./configure --prefix=/opt/local
```

Read the documentation to be sure!

You mean we might see “errors”?

- Yes, it is much more common than errors using `yum`
- Errors may occur at *any* point:

  configure, build, install, run

Troubleshooting, step 1

- Read the documentation
- There may be a “troubleshooting” section
- There may be some known issues
  - and ways to fix them

Troubleshooting configuration errors

- These errors are *your friends*
- Typical cause: a required library or tool is missing
- Configure errors *should* make this clear

Solutions if the library is not installed

1. Download and install the library
    - Either as a package or "by hand"
2. May be able to disable features that require the missing library

But what if the library is installed?

- The library is not where `configure` expects
    - This error prevented later build or run-time problems
- See if there is a `configure` option to specify library locations
- Hack: make a symbolic link to where the library is expected

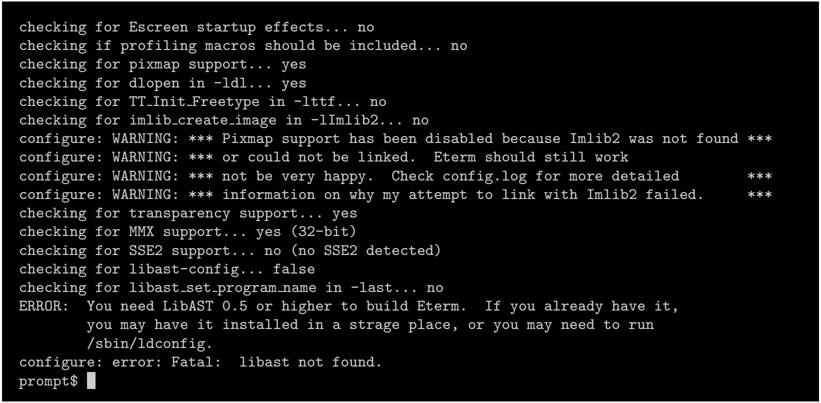
Example ./configure errors

```
prompt$
```

Example ./configure errors

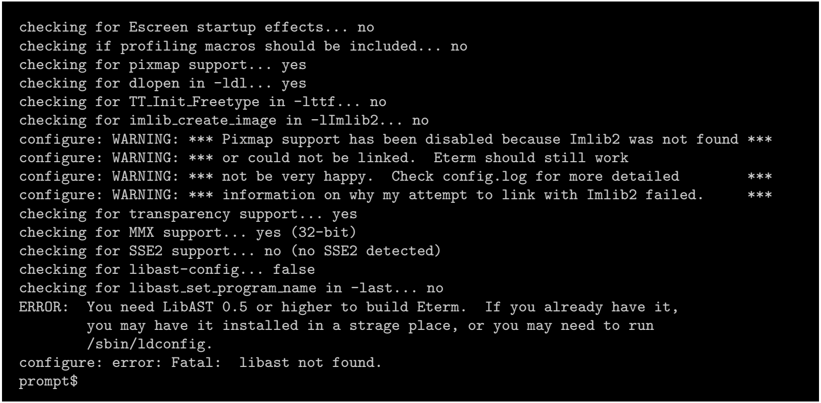
```
prompt$ ./configure
```

Example ./configure errors



```
checking for Escreen startup effects... no
checking if profiling macros should be included... no
checking for pixmap support... yes
checking for dlopen in -ldl... yes
checking for TT_Init_Freetype in -lttf... no
checking for imlib_create_image in -lImlib2... no
configure: WARNING: *** Pixmap support has been disabled because Imlib2 was not found ***
configure: WARNING: *** or could not be linked.  Eterm should still work
configure: WARNING: *** not be very happy.  Check config.log for more detailed        ***
configure: WARNING: *** information on why my attempt to link with Imlib2 failed.     ***
checking for transparency support... yes
checking for MMX support... yes (32-bit)
checking for SSE2 support... no (no SSE2 detected)
checking for libast-config... false
checking for libast_set_program_name in -last... no
ERROR:  You need LibAST 0.5 or higher to build Eterm.  If you already have it,
        you may have it installed in a strage place, or you may need to run
        /sbin/ldconfig.
configure: error: Fatal:  libast not found.
prompt$ █
```

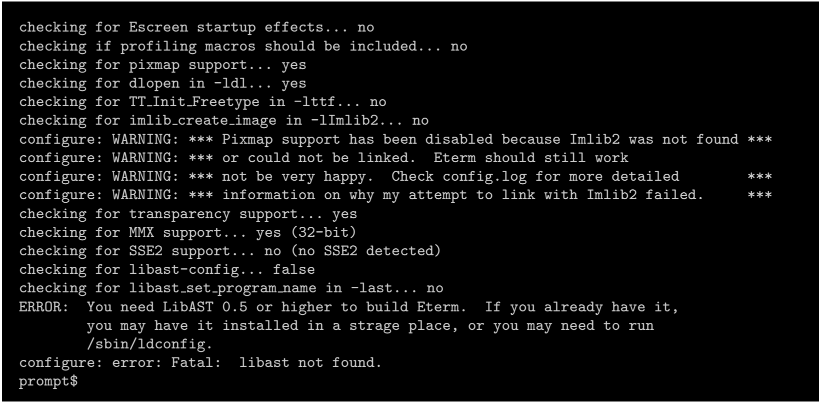
Example ./configure errors



```
checking for Escreen startup effects... no
checking if profiling macros should be included... no
checking for pixmap support... yes
checking for dlopen in -ldl... yes
checking for TT_Init_Freetype in -lttf... no
checking for imlib_create_image in -lImlib2... no
configure: WARNING: *** Pixmap support has been disabled because Imlib2 was not found ***
configure: WARNING: *** or could not be linked.  Eterm should still work
configure: WARNING: *** not be very happy.  Check config.log for more detailed       ***
configure: WARNING: *** information on why my attempt to link with Imlib2 failed.    ***
checking for transparency support... yes
checking for MMX support... yes (32-bit)
checking for SSE2 support... no (no SSE2 detected)
checking for libast-config... false
checking for libast_set_program_name in -last... no
ERROR:  You need LibAST 0.5 or higher to build Eterm.  If you already have it,
        you may have it installed in a strage place, or you may need to run
        /sbin/ldconfig.
configure: error: Fatal:  libast not found.
prompt$
```

- You will need to install `libast` first

Example ./configure errors



```
checking for Escreen startup effects... no
checking if profiling macros should be included... no
checking for pixmap support... yes
checking for dlopen in -ldl... yes
checking for TT_Init_Freetype in -lttf... no
checking for imlib_create_image in -lImlib2... no
configure: WARNING: *** Pixmap support has been disabled because Imlib2 was not found ***
configure: WARNING: *** or could not be linked.  Eterm should still work
configure: WARNING: *** not be very happy.  Check config.log for more detailed       ***
configure: WARNING: *** information on why my attempt to link with Imlib2 failed.     ***
checking for transparency support... yes
checking for MMX support... yes (32-bit)
checking for SSE2 support... no (no SSE2 detected)
checking for libast-config... false
checking for libast_set_program_name in -last... no
ERROR:  You need LibAST 0.5 or higher to build Eterm.  If you already have it,
        you may have it installed in a strage place, or you may need to run
        /sbin/ldconfig.
configure: error: Fatal:  libast not found.
prompt$
```

- You will need to install `libast` first
- Note the warning about `Imlib2`
  - You should probably install that, also

Troubleshooting build errors

- These errors can be tough to decipher
- These errors can be tough to fix
- May be caused by missing library
  - It can be *much* less obvious, “which one”
- May be caused by path problems
- May in fact be caused by errors in the C code
  - . . . which you are, of course, free to try and fix

Build error example (1)

From the previous example:

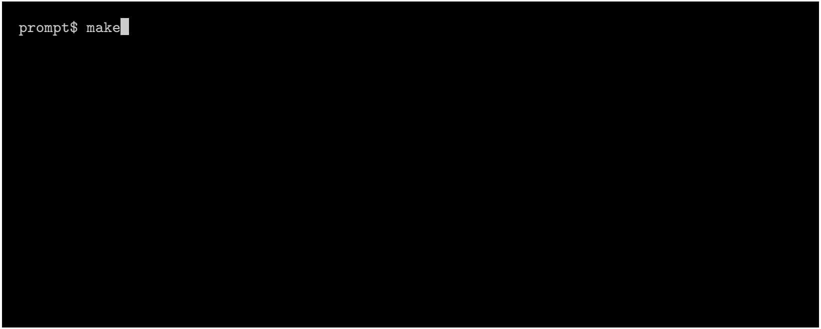
1. Install `libast`
2. Run `./configure` again
3. Remove `libast`
4. Try to build and see what happens

```
prompt$
```

Build error example (1)

From the previous example:

1. Install `libast`
2. Run `./configure` again
3. Remove `libast`
4. Try to build and see what happens

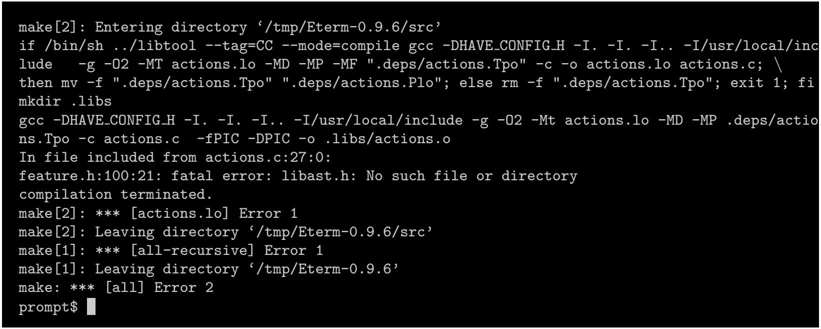


```
prompt$ make
```

Build error example (1)

From the previous example:

1. Install `libast`
2. Run `./configure` again
3. Remove `libast`
4. Try to build and see what happens



```
make[2]: Entering directory ‘/tmp/Eterm-0.9.6/src’
if /bin/sh ../libtool --tag=CC --mode=compile gcc -DHAVE_CONFIG_H -I. -I. -I.. -I/usr/local/inc
lude   -g -O2 -MT actions.lo -MD -MP -MF ".deps/actions.Tpo" -c -o actions.lo actions.c; \
then mv -f ".deps/actions.Tpo" ".deps/actions.Plo"; else rm -f ".deps/actions.Tpo"; exit 1; fi
mkdir .libs
gcc -DHAVE_CONFIG_H -I. -I. -I.. -I/usr/local/include -g -O2 -Mt actions.lo -MD -MP .deps/actio
ns.Tpo -c actions.c  -fPIC -DPIC -o .libs/actions.o
In file included from actions.c:27:0:
feature.h:100:21: fatal error: libast.h: No such file or directory
compilation terminated.
make[2]: *** [actions.lo] Error 1
make[2]: Leaving directory ‘/tmp/Eterm-0.9.6/src’
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory ‘/tmp/Eterm-0.9.6’
make: *** [all] Error 2
prompt$ █
```

Build error example (2)

From the previous example:

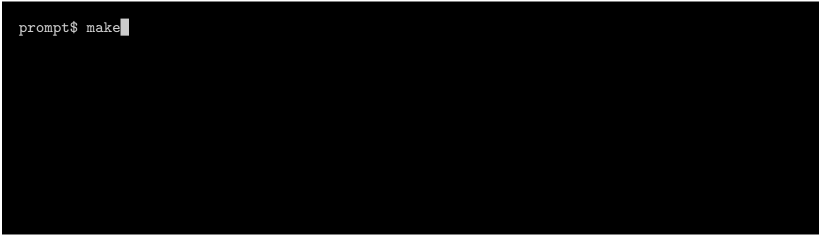
1. Reinstall `libast`
2. Try to build and see what happens

```
prompt$
```

Build error example (2)

From the previous example:

1. Reinstall `libast`
2. Try to build and see what happens



```
prompt$ make
```

Build error example (2)

From the previous example:

1. Reinstall `libast`
2. Try to build and see what happens

```
gcc -g -O2 -o .libs/Eterm main.o -L/usr/local/lib ./.libs/libEterm.so /usr/local/lib/libast.so
 -lfreetype -lSM -lICE -ldl -lXext -lX11 -lutil -lm -Wl,--rpath -Wl,/usr/local/lib -Wl,--rpath
 -Wl,/usr/local/lib/usr/local/lib/Eterm
./.libs/libEterm.so:  undefined reference to ‘imlib_free_pixmap_and_mask’
collect2:  ld returned 1 exit status
make[2]: *** [Eterm] Error 1
make[2]: Leaving directory ‘/tmp/Eterm-0.9.6/src’
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory ‘/tmp/Eterm-0.9.6’
make: *** [all] Error 2
prompt$ █
```

Build error example (2)

From the previous example:

1. Reinstall `libast`
2. Try to build and see what happens

```
gcc -g -O2 -o .libs/Eterm main.o -L/usr/local/lib ./.libs/libEterm.so /usr/local/lib/libast.so
 -lfreetype -lSM -lICE -ldl -lXext -lX11 -lutil -lm -Wl,--rpath -Wl,/usr/local/lib -Wl,--rpath
 -Wl,/usr/local/lib/usr/local/lib/Eterm
./.libs/libEterm.so:  undefined reference to ‘imlib_free_pixmap_and_mask’
collect2:  ld returned 1 exit status
make[2]: *** [Eterm] Error 1
make[2]: Leaving directory ‘/tmp/Eterm-0.9.6/src’
make[1]: *** [all-recursive] Error 1
make[1]: Leaving directory ‘/tmp/Eterm-0.9.6’
make: *** [all] Error 2
prompt$
```

Remember the warning about `Imlib2`?

Install errors

- Did you forget to do this as `root`?
- Did you try to install to a path that does not exist?

High-level package management

apt : Debian’s package system

dnf : Replacement for yum

yum : Fedora/Red Hat package system

Low-level package management

dpkg : Debian Package; for `.deb` files

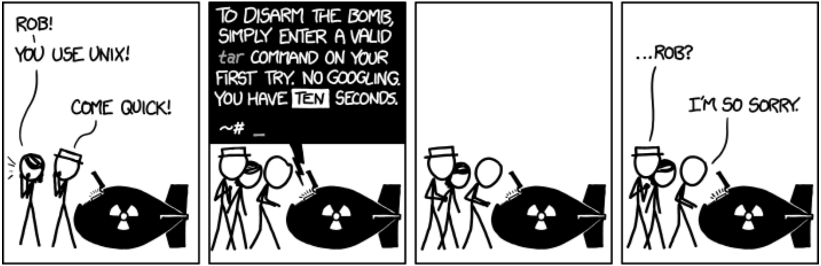
rpm : Red Had Package Manager; for `.rpm` files

General utilities, useful for installing from source

diff	:	show file differences
make	:	build something
patch	:	apply changes to a file
rsync	:	remote file copy
tar	:	manage archive files
unzip	:	unpack a “zip” archive
zip	:	pack a “zip” archive

Read the documentation

An appropriate xkcd comic: `http://xkcd.com/1168`


ROB!
YOU USE UNIX!
COME QUICK!
TO DISARM THE BOMB, SIMPLY ENTER A VALID tar COMMAND ON YOUR FIRST TRY. NO GOOGLING. YOU HAVE TEN SECONDS.
~# _
...ROB?
I'M SO SORRY.

End of lecture