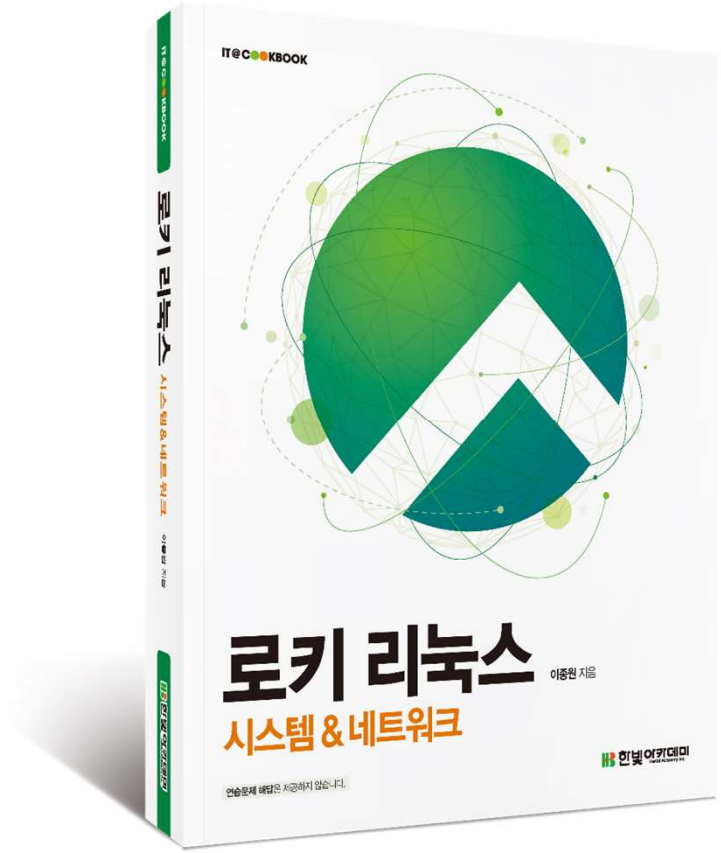


Chapter 08 소프트웨어 관리

목차

학습목표

- RPM 패키지를 설치하거나 삭제할 수 있다.
- dnf를 이용하여 패키지를 설치하거나 삭제할 수 있다.
- 파일을 묶어 아카이브 파일을 만들고 압축할 수 있다.
- 프로그램을 스스로 컴파일할 수 있다.
- makefile을 작성하고 make 명령으로 실행 파일을 만들 수 있다.

00 Preview

00 Preview

8장의 내용 구성

- 패키지의 의존성을 자동으로 해결하면서 패키지를 설치해 주는 관리 도구인 dnf가 등장
- dnf는 기존의 yum을 대체한 것으로, 네트워크만 연결 되어 있으면 자동으로 dnf 저장소의 정보를 확인하기 때문에 패키지의 의존성 문제를 해결하면서 설치할 수 있음
- 모든 패키지가 바이너리 형태로 제공되는 것은 아니며, 소스 코드로 배포되는 패키지를 다운로드하여 설치해야 할 때 소스 코드를 묶어 하나의 아카이브 파일로 만들고 이를 압축하여 배포함

01 RPM 패키지 설치

01 RPM 패키지 설치

RPM

- 대표적인 리눅스 회사인 레드햇에서 만든 패키지 관리 도구로 파일을 쉽게 설치할 수 있음

RPM의 특징

- 바이너리 파일로 구성되어 있어 컴파일이 필요 없다
- 패키지의 파일들이 관련 디렉터리로 바로 설치된다
- 한 번에 설치된 패키지의 파일을 일괄적으로 삭제할 수 있다
- 기존에 설치된 패키지를 삭제하지 않고 바로 업그레이드할 수 있다
- 패키지의 설치 상태를 검증할 수 있다
- 패키지에 대한 정보를 제공한다

01 RPM 패키지 설치

■ RPM 패키지의 이름 구성

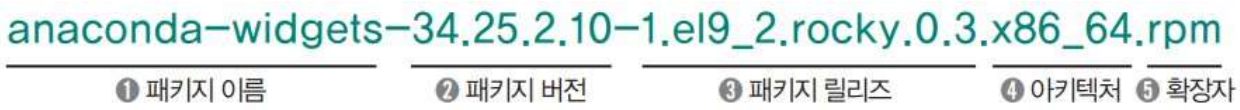


그림 8-1 RPM 패키지의 이름 구성

① 패키지 이름: 첫 번째 항목에 패키지의 이름이 나옴

② 패키지 버전: 두 번째 항목에 패키지의 버전이 나옴

③ 패키지 릴리즈: 릴리즈는 해당 버전에서 몇 번째 만들어진 것인지를 나타냄.
'1.el9_2'에서 1은 첫 번째 만들어진 패키지임을 뜻하고, el9_2는 CentOS 9용으로 만들어진 것임을 의미.
로키 리눅스는 CentOS의 후속 프로젝트로 탄생한 소프트웨어이므로 CentOS와 당연히 호환성을 유지함.
rocky.0.3은 이 패키지가 로키 리눅스용으로 릴리즈된 것임을 의미함

01 RPM 패키지 설치

RPM 패키지의 이름 구성

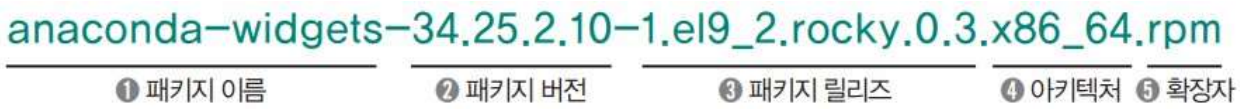


그림 8-1 RPM 패키지의 이름 구성

④ 아키텍처: 사용하는 시스템 아키텍처를 의미함.
x86은 인텔 x86 CPU를 뜻하고 64는 64비트 운영체임을 의미함

⑤ 확장자: RPM 패키지의 확장자로 .rpm을 사용하여 다른 패키지와 구분함

01 RPM 패키지 설치

RPM 패키지 명령

rpm

- 기능 RPM 패키지를 관리한다.
  --quiet: 최대한 출력을 줄이고 오류 메시지만 출력한다.
  --help: 도움말을 출력한다.
  --version: 사용중인 rpm의 버전을 출력한다.

- 현재 사용 중인 rpm의 버전을 알아보기

```
[user1@localhost ~]$ rpm --version
RPM 버전 - 4.16.1.3
```

01 RPM 패키지 설치

RPM 패키지 설치와 업그레이드

- rpm 명령으로 패키지를 설치할 때 일반적인 형식

rpm -i [설치옵션] [패키지명]

- 기능 RPM 패키지를 설치한다.
- 설치옵션 -h: #(해시)를 출력한다.
  -v: 설치 과정에 대한 메시지를 출력한다.
  --replacefiles: 이미 설치된 다른 패키지의 파일을 덮어쓰면서라도 패키지를 강제로 설치한다.
  --replacepkgs: 패키지가 이미 설치되어 있어도 다시 설치한다.
  --test: 설치하지는 않고 충돌 사항이 있는지만 점검하고 보고한다.

- rpm 명령으로 패키지를 업그레이드하여 설치할 때 일반적인 형식

rpm -U [설치옵션] [패키지명]

- 기능 RPM 패키지를 업그레이드한다.
- 설치옵션 -h: #를 출력한다.
  -v: 설치 과정에 대한 메시지를 출력한다.
  -U: 패키지를 업그레이드하여 설치한다.

01 RPM 패키지 설치

RPM 패키지 설치와 업그레이드

- 패키지를 업그레이드할 때는 -Uvh 옵션을 사용. -U 옵션이 대문자임에 주의
- -U 옵션으로 패키지를 설치하면 기존에 설치된 패키지는 업그레이드하여 설치하고, 새로운 패키지는 그대로 설치해 줌
- 보통 -i 옵션보다 -U 옵션을 더 많이 사용

01 RPM 패키지 설치

RPM 패키지 다운로드하기

- 패키지 설치를 실습하려면 로키 리눅스 패키지가 필요
- 로키 리눅스를 설치할 때 사용한 iso 파일에서는 RPM 패키지를 찾을 수 없고, 네트워크를 통해 RPM 패키지를 다운로드해야 함
- RPM 패키지는 http://pkgs.org 또는 http://rpmfind.net에서 다운로드할 수 있는데, 여기서는 pkgs.org 사이트를 이용
- [현재 활동]을 선택하여 데스크바를 띄우고 파이어폭스 웹 브라우저를 동작시켜 http://pkgs.org에 접속함
- http://pkgs.org 사이트에서는 리눅스 배포판별로 패키지를 구분하여 제공하고 있음
  이 사이트의 하단으로 이동하면 로키 리눅스 9와 8 배포판이 있음

01 RPM 패키지 설치

■ RPM 패키지 다운로드하기

- 로키 리눅스 배포판의 세부 목록을 살펴보면 다수의 패키지가 CentOS 용으로 구성되어 있음
  이는 로키 리눅스가 출시된 지 얼마 되지 않아 아직 과도기적 상태이기 때문

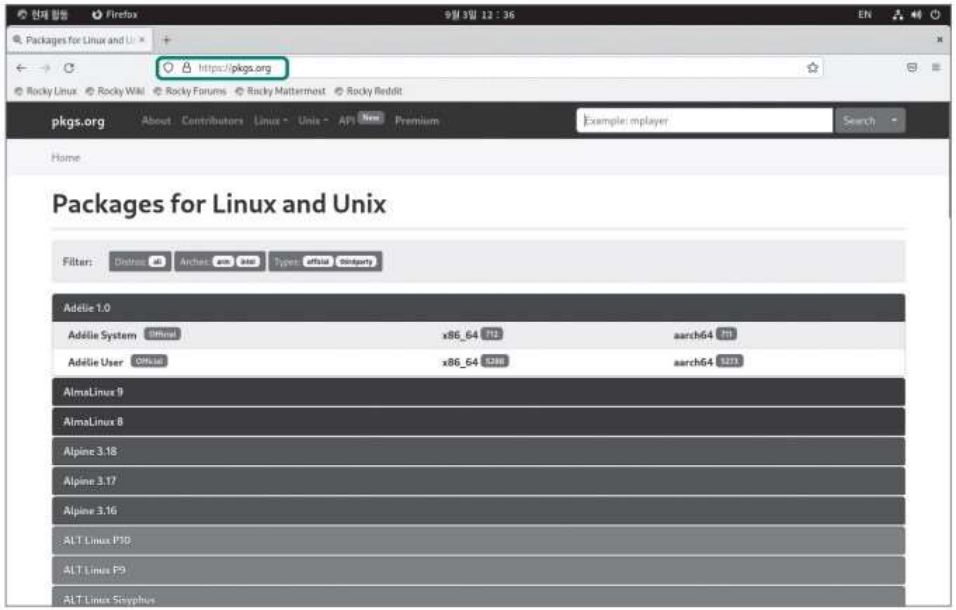


그림 8-2 파이어폭스에서 pkgs.org 접속 화면

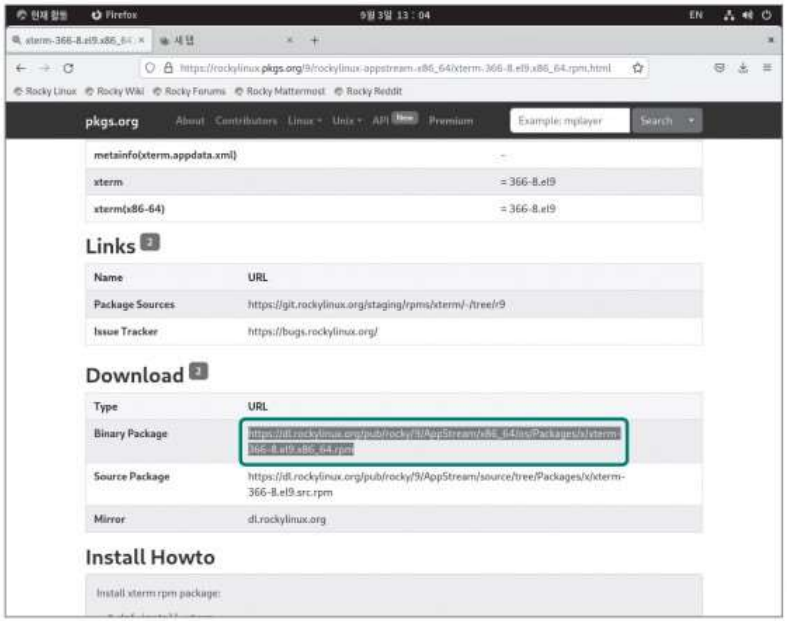
01 RPM 패키지 설치

RPM 패키지 다운로드하기

- 터미널 에뮬레이터인 xterm을 설치해 보기
- pkgs.org에서 xterm을 검색하면 여러 버전이 나오는데 로키 리눅스 9에 있는 버전을 선택
- 검색 결과 중 xterm-366-8.el9.x86_64. rpm을 선택하면 패키지에 대한 자세한 설명이 나옴
- 브라우저의 하단으로 스크롤하면 다운로드 항목이 나오는데, 바이너리 패키지의 링크를 복사하여 주소창에 붙이기를 하면 rpm 파일이 다운로드됨

01 RPM 패키지 설치

■ RPM 패키지 다운로드하기



(a) xterm 다운로드 주소 복사하기

(b) 주소창에 붙이기

그림 8-3 xterm 다운로드

01 RPM 패키지 설치

RPM 패키지 다운로드하기

- 다운로드한 xterm 패키지 파일은 사용자 홈 디렉터리의 다운로드 디렉터리에 저장됨

```
[user1@localhost ~]$ ls 다운로드
xterm-366-8.el9.x86_64.rpm
```

01 RPM 패키지 설치

■ 패키지 설치와 의존성

- 패키지를 설치하는 -ivh 옵션을 사용함. 이 옵션을 사용하면 앞에서 설치 과정에 대한 메시지와 #를 출력하여 사용자가 진행 상황을 알 수 있도록 해줌
- root 계정으로 전환하고 다운로드 디렉터리로 이동하여 설치 명령을 실행함

```
[user1@localhost ~]$ su -
암호:
[root@localhost ~]# cd ~user1/다운로드
[root@localhost 다운로드]# rpm -ivh xterm-366-8.el9.x86_64.rpm
경고: xterm-366-8.el9.x86_64.rpm: Header V4 RSA/SHA256 Signature, key ID 350d275d:
NOKEY
오류: Failed dependencies:
        libXaw.so.7()(64bit) is needed by xterm-366-8.el9.x86_64
        xterm-resize = 366-8.el9 is needed by xterm-366-8.el9.x86_64
```

01 RPM 패키지 설치

■ 패키지 설치와 의존성

- xterm 패키지를 설치하려면 libXaw.so.7과 xterm-resize가 필요하다는 오류 메시지가 출력되었음
- A라는 패키지를 설치하려면 B라는 패키지가 필요하다는 것이 패키지 의존성
- 이 경우에는 먼저 libXaw와 xterm-resize 패키지를 설치해야 함
- 이 두 패키지를 앞에서와 같은 방법으로 다운로드하여 설치
- libXaw와 xterm-resize 패키지가 설치되었음

```
[root@localhost 다운로드]# rpm -ivh libXaw-1.0.13-19.el9.x86_64.rpm
경고: libXaw-1.0.13-19.el9.x86_64.rpm: Header V4 RSA/SHA256 Signature, key ID 350d275d:
NOKEY
Verifying...                          ################################# [100%]
준비 중...                            ################################# [100%]
Updating / installing...
   1:libXaw-1.0.13-19.el9             ################################# [100%]
[root@localhost 다운로드]# rpm -ivh xterm-resize-366-8.el9.x86_64.rpm
경고: xterm-resize-366-8.el9.x86_64.rpm: Header V4 RSA/SHA256 Signature, key ID
350d275d: NOKEY
Verifying...                          ################################# [100%]
준비 중...                            ################################# [100%]
Updating / installing...
   1:xterm-resize-366-8.el9           ################################# [100%]
```

01 RPM 패키지 설치

■ 패키지 설치와 의존성

- xterm 패키지를 설치해 보자. 이번에는 -Uvh 옵션을 사용함

```
[root@localhost 다운로드]# rpm -Uvh xterm-366-8.el9.x86_64.rpm
경고: xterm-366-8.el9.x86_64.rpm: Header V4 RSA/SHA256 Signature, key ID 350d275d:
NOKEY
Verifying...                          ################################# [100%]
준비 중...                            ################################# [100%]
Updating / installing...
   1:xterm-366-8.el9                  ################################# [100%]
```

- xterm 패키지가 설치됨

01 RPM 패키지 설치

■ 패키지 설치와 의존성

- xterm 은 X윈도 환경에서 터미널을 에뮬레이션해 주는 프로그램으로, 실행해 보면 [그림 8-4]와 같이 동작함. xterm은 일반 사용자 계정에서 실행함

그림 8-4 xterm 동작 화면

01 RPM 패키지 설치

RPM 패키지 정보 검색

- rpm 명령으로 패키지의 정보를 검색할 때는 -q 옵션을 사용

rpm -q [질의옵션]

- 기능 RPM 패키지의 정보를 검색한다.
- 질의옵션 -a: 전체 패키지의 목록을 출력한다.
  -f 파일명: 파일명을 포함하고 있는 패키지명을 출력한다.
  -p 패키지명: 지정한 패키지의 상세 정보를 출력한다.
  -i: 자세한 정보를 출력한다.
  -R: 의존하고 있는 패키지의 목록을 출력한다.
  -l: 패키지 내의 파일을 출력한다.
  -s: 패키지 안에 들어 있는 파일의 상태를 출력한다.
  -d: 문서 파일만 출력한다.
  -c: 설정 파일만 출력한다.

01 RPM 패키지 설치

■ 전체 패키지 목록 출력하기: -qa 옵션

- rpm -qa 명령은 현재 설치된 전체 패키지의 목록을 출력함

```
[root@localhost 다운로드]# rpm -qa | more
libgcc-11.3.1-4.3.el9.x86_64
fonts-filesystem-2.0.5-7.el9.1.noarch
linux-firmware-whence-20230310-134.el9_2.noarch
crypto-policies-20221215-1.git9a18988.el9.noarch
hwdata-0.348-9.7.el9.noarch
xkeyboard-config-2.33-2.el9.noarch
(생략)
```

01 RPM 패키지 설치

■ 특정 패키지 설치 여부 확인하기: -qa | grep

- rpm -qa 명령과 grep 명령을 함께 사용하면 특정 패키지가 설치되어 있는지 확인할 수 있음
- 패키지 목록 중 bash와 관련된 패키지가 있는지 검색한 예

```
[root@localhost 다운로드]# rpm -qa | grep bash
bash-5.1.8-6.el9_1.x86_64
bash-completion-2.11-4.el9.noarch
```

■ 특정 파일을 포함하는 패키지 확인하기: -qf 옵션

- 특정 파일이 어떤 패키지에 속해 있는지 궁금할 때는 f 옵션을 사용
- f 옵션으로 파일을 지정할 때는 다음과 같이 절대 경로로 지정해야 함.
  검색 결과를 보면 ls 명령이 coreutils 패키지에 속해 있음을 알 수 있음

```
[root@localhost 다운로드]# rpm -qf /usr/bin/ls
coreutils-8.32-34.el9.x86_64
```

01 RPM 패키지 설치

■ 패키지의 상세 정보 확인하기: -qi 옵션

- 특정 패키지의 상세 정보를 확인하려면 i 옵션을 사용해야 하는데 -qi 옵션만 사용할 때는 패키지 이름을 지정함
- 이때 패키지 이름은 패키지 이름 구성에서 첫 번째 항목에 해당하는 부분을 지정하면 됨

```
[root@localhost 다운로드]# rpm -qi bash
Name        : bash
Version     : 5.1.8
Release     : 6.el9_1
Architecture: x86_64
Install Date: 2023년 07월 09일 (일) 오전 10시 39분 31초
Group       : Unspecified
Size        : 7738634
License     : GPLv3+
Signature   : RSA/SHA256, 2023년 01월 24일 (화) 오전 07시 49분 22초, Key ID 702d426d350d275d
Source RPM  : bash-5.1.8-6.el9_1.src.rpm
Build Date  : 2023년 01월 24일 (화) 오전 07시 43분 46초
(생략)
```

01 RPM 패키지 설치

■ 패키지의 상세 정보 확인하기: -qip와 -qif 옵션

- 특정 패키지의 상세 정보를 확인하기 위해 i 옵션을 사용할 때 p나 f 옵션을 함께 사용하여 지정한 패키지 또는 지정한 파일이 속한 패키지에 대한 상세 정보를 출력할 수 있음
- 먼저 f 옵션을 사용할 경우, 다음과 같이 해당 파일이 속한 패키지의 상세 정보가 출력됨

```
[root@localhost 다운로드]# rpm -qif /usr/bin/ls
Name        : coreutils
Version     : 8.32
Release     : 34.el9
Architecture: x86_64
Install Date: 2023년 07월 09일 (일) 오전 10시 39분 49초
(생략)
Packager    : Rocky Linux Build System (Peridot) <releng@rockylinux.org>
Vendor      : Rocky Enterprise Software Foundation
(생략)
```

01 RPM 패키지 설치

■ 패키지의 상세 정보 확인하기: -qip와 -qif 옵션

- p 옵션을 사용할 때는 해당 패키지가 위치한 경로를 지정하거나 디렉터리로 이동해야 함
- 패키지명만 지정하면 오류가 발생함
- 패키지 파일이 위치한 디렉터리에서 패키지명을 지정하면 패키지의 상세 정보를 출력함

```
[root@localhost 다운로드]# rpm -qip xterm-366-8.el9.x86_64.rpm
경고: xterm-366-8.el9.x86_64.rpm: Header V4 RSA/SHA256 Signature, key ID 350d275d:
NOKEY
Name        : xterm
Version     : 366
Release     : 8.el9
Architecture: x86_64
(생략)
```

01 RPM 패키지 설치

■ 패키지가 설치한 파일 목록 확인하기: -ql 옵션

- 특정 패키지가 설치한 파일의 목록을 확인하려면 l 옵션을 사용함
- xterm 패키지가 설치한 파일의 목록을 출력한 예. 생각보다 많은 파일이 설치되었음을 알 수 있음

```
[root@localhost 다운로드]# rpm -ql xterm
/usr/bin/koi8rxterm
/usr/bin/uxterm
/usr/bin/xterm
/usr/lib/.build-id
/usr/lib/.build-id/d1
(생략)
```

01 RPM 패키지 설치

■ 패키지의 의존성 확인하기: -qR 옵션

- 앞에서 xterm 패키지를 설치할 때 의존성이 있는 패키지를 먼저 설치해야 했음
- 특정 패키지가 어떤 파일에 의존성이 있는지 알아보려면 R 옵션을 사용함
- xterm 패키지의 의존성을 알아본 예

```
[root@localhost 다운로드]# rpm -qR xterm
/usr/bin/sh
libICE.so.6()(64bit)
libX11.so.6()(64bit)
libXaw.so.7()(64bit)
(생략)
```

01 RPM 패키지 설치

RPM 패키지 삭제

rpm -e [패키지명]

- 기능 RPM 패키지를 삭제한다.

- 앞에서 설치한 xterm 패키지를 삭제하면 다음과 같음
- -qa 옵션으로 확인해 보면 설치된 패키지 목록에 xterm은 없고 xterm-resize만 있음

```
[root@localhost 다운로드]# rpm -e xterm
[root@localhost 다운로드]# rpm -qa | grep xterm
xterm-resize-366-8.el9.x86_64
```

01 RPM 패키지 설치

■ 패키지 정보 확인하고 설치하기

① 파이어폭스에서 pkgs.org로 접속

② pkgs.org에서 gnome-2048을 검색

③ 아직 로키 리눅스용 버전이 없으므로, Fedora 38버전 ggnome-2048-3.38.2-7.fc38.x86_64.rpm 패키지를 다운로드

④ 설치할 패키지의 상세 정보를 확인

```
[root@localhost 다운로드]# rpm -qip gnome-2048-3.38.2-7.fc38.x86_64.rpm
경고: gnome-2048-3.38.2-7.fc38.x86_64.rpm: Header V4 RSA/SHA256 Signature, key ID
eb10b464: NOKEY
Name        : gnome-2048
Version     : 3.38.2
Release     : 7.fc38
Architecture: x86_64
(생략)
```

01 RPM 패키지 설치

■ 패키지 정보 확인하고 설치하기

⑤ 패키지를 설치

```
[root@localhost 다운로드]# rpm -ivh gnome-2048-3.38.2-7.fc38.x86_64.rpm
경고: gnome-2048-3.38.2-7.fc38.x86_64.rpm: Header V4 RSA/SHA256 Signature, key ID
eb10b464: NOKEY
오류: Failed dependencies:
        libgnome-games-support-1.so.3()(64bit) is needed by gnome-2048-3.38.2-7.fc38.
x86_64
```

01 RPM 패키지 설치

■ 패키지 정보 확인하고 설치하기

⑥ 의존성이 있는 패키지를 추가로 설치해야 함. libgnome-games-support 패키지를 검색하여 다운로드해서 설치한 후, gnome-2048을 다시 설치함

```
[root@localhost 다운로드]# rpm -ivh libgnome-games-support-1.8.2-4.fc38.x86_64.rpm
경고: libgnome-games-support-1.8.2-4.fc38.x86_64.rpm: Header V4 RSA/SHA256 Signature,
key ID eb10b464: NOKEY
Verifying...                          ################################# [100%]
준비 중...                            ################################# [100%]
Updating / installing...
   1:libgnome-games-support-1.8.2-4.fc#################################### [100%]
[root@localhost 다운로드]# rpm -ivh gnome-2048-3.38.2-7.fc38.x86_64.rpm
경고: gnome-2048-3.38.2-7.fc38.x86_64.rpm: Header V4 RSA/SHA256 Signature, key ID
eb10b464: NOKEY
Verifying...                          ################################# [100%]
준비 중...                            ################################# [100%]
Updating / installing...
   1:gnome-2048-3.38.2-7.fc38         ################################# [100%]
```

01 RPM 패키지 설치

■ 패키지 정보 확인하고 설치하기

⑦ 패키지가 설치한 파일을 확인. 실행 파일의 이름을 알 수 있음

```
[root@localhost 다운로드]# rpm -ql gnome-2048
/usr/bin/gnome-2048
/usr/lib/.build-id
(생략)
```

⑧ 실행 파일이 /usr/bin/gnome-2048임을 알았다. 일반 사용자 계정에서 실행함

```
[user1@localhost ~]$ gnome-2048 &
```

01 RPM 패키지 설치

■ 패키지 정보 확인하고 설치하기

- 정상적으로 설치되었다면 [그림 8-5]와 같은 창이 실행되는 것을 확인할 수 있음

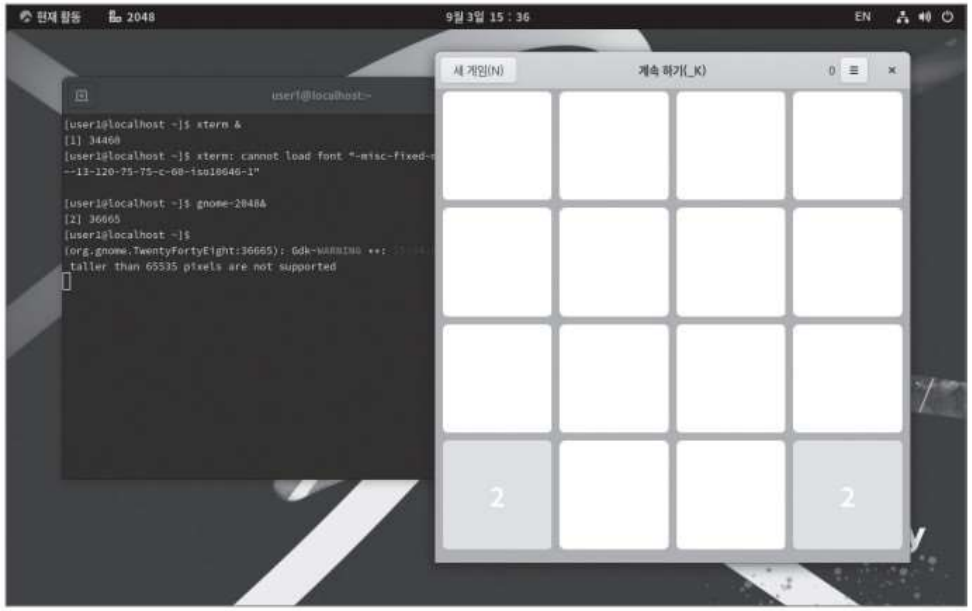


그림 8-5 gnome-2048 동작 화면

01 RPM 패키지 설치

■ 패키지 정보 확인하고 설치하기

혼자해보기 패키지 정보 확인하고 설치하기

❶ pkgs.org에서 gnome-robots를 검색한다.

❷ 로키 리눅스용 gnome-robots 패키지가 없으므로 Fedora 38에 있는 gnome-robots-40.0-5.fc38.x86_64.rpm 패키지를 다운로드하여 설치한다.

❸ gnome-robots-40.0-5.fc38.x86_64.rpm 패키지가 설치한 파일 중 실행 파일이 무엇인지 확인한다.

❹ gnome-robots-40.0-5.fc38.x86_64.rpm 패키지가 설치한 실행 파일을 실행하고 오른쪽과 같은 게임 화면인지 확인한다.

02 dnf를 이용한 패키지 설치

02 dnf를 이용한 패키지 설치

dnf의 특징

- dnf는 RPM 기반의 리눅스 패키지 관리자로, 기존 패키지 관리자인 yum을 대체 한 것.
  yum 명령을 확인해 보면 dnf로 업그레이드되었다는 설명이 나옴

```
[root@localhost 다운로드]# man yum
YUM(8)                    DNF                    YUM(8)

NAME
       yum - redirecting to DNF Command Reference

SYNOPSIS
       dnf [options] <command> [<args>...]
(생략)
```

- 기존의 yum 명령도 dnf에 대한 심볼릭 링크로 유지하고 있어 yum을 사용해도 됨

```
[root@localhost 다운로드]# ls -l /usr/bin/yum
lrwxrwxrwx. 1 root root 5  5월  9 16:56 /usr/bin/yum -> dnf-3
```

02 dnf를 이용한 패키지 설치

dnf의 특징

- dnf의 최대 장점은 패키지를 설치할 때 번거로웠던 패키지 의존성 문제를 자동으로 해결했다는 점
- dnf의 특징
  - 패키지의 의존성을 자동으로 해결하여 설치·업그레이드·삭제할 수 있다
  - 설치 가능한 패키지에 대한 정보가 담긴 저장소repository를 가지고 있다
  - 저장소의 정보에 따라 패키지를 관리한다

02 dnf를 이용한 패키지 설치

dnf의 특징

- dnf의 기본 형식은 다음과 같음

dnf

- 기능 RPM 기반의 패키지를 관리한다.
- 형식 dnf [옵션] [명령] [패키지명]
- 옵션 -h: 도움말을 출력한다.
  -y: 설치 과정의 모든 질문에 yes로 대답한다.
  -v: 자세한 메시지를 출력한다.
- 명령 install: 패키지를 설치한다.
  upgrade: 패키지를 업데이트한다.
  check-update: 패키지를 확인한다.
  search: 패키지를 검색한다.
  remove: 패키지를 삭제한다.
  list: 패키지의 목록을 확인한다.
  info: 패키지의 정보를 확인한다.
- 사용 예 dnf install xterm
  dnf list all

02 dnf를 이용한 패키지 설치

■ 패키지 목록 확인

- dnf list 명령의 출력 형식

  패키지명.아키텍처 버전-릴리즈 저장소 또는 @설치한 저장소

- 옵션 없이 dnf list만 실행하면 현재 설치된 패키지 목록과 업그레이드할 수 있는 패키지 목록이 출력. dnf list all 명령의 결과와 같음

- dnf list 명령의 실행 결과는 다음과 같음

```
[root@localhost 다운로드]# dnf list | more
마지막 메타자료 만료확인(0:32:58 이전): 2023년 09월 03일 (일) 오후 03시 32분 39초.
설치된 꾸러미
ModemManager.x86_64                    1.20.2-1.el9          @anaconda
ModemManager-glib.x86_64               1.20.2-1.el9          @anaconda
NetworkManager.x86_64                  1:1.42.2-1.el9        @anaconda
(생략)
```

02 dnf를 이용한 패키지 설치

■ 패키지 목록 확인

- dnf list 명령에서 사용할 수 있는 옵션

표 8-1 dnf list 명령의 옵션

형식	기능
dnf list --all	이미 설치되었거나 설치 가능한 모든 패키지 목록을 출력한다.
dnf list --available	dnf 저장소에서 설치 가능한 모든 패키지 목록을 출력한다.
dnf list --upgrades	dnf 저장소에서 업그레이드가 가능한 패키지 목록을 출력한다.
dnf list --installed	이미 설치된 패키지 목록을 출력한다.
dnf list --installed 패키지명	지정한 패키지가 설치되어 있는지 확인한다.

02 dnf를 이용한 패키지 설치

dnf list --available

- dnf 저장소에 설치 가능한 패키지들의 목록을 보려면 available 옵션을 지정함
- 명령의 실행 결과는 다음과 같음. 실행 결과에서 appstream과 baseos는 저장소를 의미함
- baseos 저장소에는 운영체제의 핵심 기능과 관련된 패키지가 있고, appstream 저장소에는 부가적인 패키지와 모듈이 있음

```
[root@localhost 다운로드]# dnf list --available
마지막 메타자료 만료확인(0:36:00 이전): 2023년 09월 03일 (일) 오후 03시 32분 39초.
사용 가능한 꾸러미
389-ds-base.x86_64                  2.2.4-3.el9                 appstream
389-ds-base-libs.x86_64             2.2.4-3.el9                 appstream
Box2D.i686                          2.4.1-7.el9                 appstream
Box2D.x86_64                        2.4.1-7.el9                 appstream
CUnit.i686                          2.1.3-25.el9                appstream
CUnit.x86_64                        2.1.3-25.el9                appstream
(생략)
ModemManager-glib.i686              1.20.2-1.el9                baseos
NetworkManager.x86_64               1:1.42.2-6.el9_2            baseos
(생략)
```

02 dnf를 이용한 패키지 설치

dnf list --upgrades

- dnf 저장소에 들어 있는 패키지 중에서 업그레이드할 수 있는 패키지의 목록을 보려면 upgrades 옵션을 지정함. 이 옵션은 dnf check-update 명령과 같은 기능을 수행함

```
[root@localhost 다운로드]# dnf list --upgrades
마지막 메타자료 만료확인(0:37:30 이전): 2023년 09월 03일 (일) 오후 03시 32분 39초.
사용 가능한 최신화
NetworkManager.x86_64                1:1.42.2-6.el9_2          baseos
NetworkManager-adsl.x86_64           1:1.42.2-6.el9_2          baseos
NetworkManager-bluetooth.x86_64      1:1.42.2-6.el9_2          baseos
NetworkManager-libnm.x86_64          1:1.42.2-6.el9_2          baseos
NetworkManager-team.x86_64           1:1.42.2-6.el9_2          baseos
NetworkManager-tui.x86_64            1:1.42.2-6.el9_2          baseos
NetworkManager-wifi.x86_64           1:1.42.2-6.el9_2          baseos
NetworkManager-wwan.x86_64           1:1.42.2-6.el9_2          baseos
alsa-sof-firmware.noarch             2.2.5-2.el9_2             appstream
(생략)
```

02 dnf를 이용한 패키지 설치

dnf list --installed

- 현재 시스템에 설치된 패키지의 목록을 출력할 때는 --installed 옵션을 지정
- dnf list --installed 명령의 실행 예는 생략

02 dnf를 이용한 패키지 설치

■ 패키지 설치

- dnf로 패키지를 설치할 때는 'dnf install 패키지명'의 형식을 사용함. 이때 의존성이 있는 패키지가 자동으로 함께 설치됨
- dnf 명령은 패키지를 설치하기 위해 우선 dnf 저장소를 검색하는데, 한 저장소에서 패키지를 찾지 못하면 자동으로 다른 미러 사이트를 검색하여 패키지를 찾음

02 dnf를 이용한 패키지 설치

■ 패키지 설치

- dnf로 패키지를 설치할 때 먼저 의존성이 있는 패키지의 존재 여부를 확인하고 함께 설치해야 할 패키지 목록을 보여줌
- 다운로드한 파일의 크기와 실제로 설치되었을 때 필요한 디스크 용량을 알려주고 계속 진행할 것인지 물어봄

```
[root@localhost 다운로드]# dnf install xterm
마지막 메타자료 만료확인(0:41:05 이전): 2023년 09월 03일 (일) 오후 03시 32분 39초.
종속성이 해결되었습니다.
================================================================================
 꾸러미                  구조          버전                 저장소          크기
================================================================================
설치 중:
 xterm                  x86_64        366-8.el9            appstream       603 k
종속 꾸러미 설치 중:
 mkfontscale            x86_64        1.2.1-3.el9          appstream       31 k
취약한 종속 꾸러미 설치 중:
 xorg-x11-fonts-misc    noarch        7.5-33.el9           appstream       5.2 M

연결 요약
================================================================================
설치  3 꾸러미

전체 내려받기 크기: 5.8 M
설치된 크기 : 7.9 M
진행할까요? [y/N]:
```

02 dnf를 이용한 패키지 설치

■ 패키지 설치

- 이때 N이라고 대답하면 설치가 종료되고, y라고 대답하면 설치가 진행됨
- 여기서는 y를 입력하여 패키지 설치를 진행해 봄

```
진행할까요? [y/N]: y
꾸러미 내려받기 중:
(1/3): mkfontscale-1.2.1-3.el9.x86_64.rpm              13 kB/s |  31 kB     00:02
(2/3): xterm-366-8.el9.x86_64.rpm                      242 kB/s | 603 kB     00:02
(3/3): xorg-x11-fonts-misc-7.5-33.el9.noarch.rpm       725 kB/s | 5.2 MB     00:07
--------------------------------------------------------------------------------
합계                                                   651 kB/s | 5.8 MB     00:09
Rocky Linux 9 - AppStream                              1.7 MB/s | 1.7 kB     00:00
(생략)
설치되었습니다:
  mkfontscale-1.2.1-3.el9.x86_64    xorg-x11-fonts-misc-7.5-33.el9.noarch    xterm-366-8.
el9.x86_64

완료되었습니다!
```

02 dnf를 이용한 패키지 설치

■ 패키지 업그레이드

- 이미 설치된 패키지를 dnf로 업그레이드할 때는 'dnf upgrade 패키지명' 형식으로 입력
- 패키지명을 지정하지 않고 'dnf upgrade'만 입력하면 현재 설치된 패키지에서 업그레이드가 필요한 전체 패키지에 대해 업그레이드를 진행하므로 주의해야 함

02 dnf를 이용한 패키지 설치

■ 패키지 업그레이드

- 앞서 dnf list --upgrades 명령으로 확인한 file.x86_64 패키지의 업그레이드를 진행해 봄

```
[root@localhost 다운로드]# dnf upgrade file.x86_64
마지막 메타자료 만료확인(0:45:06 이전): 2023년 09월 03일 (일) 오후 03시 32분 39초.
종속성이 해결되었습니다.
================================================================================
 꾸러미                  구조        버전                  저장소          크기
================================================================================
향상 중:
 file                    x86_64      5.39-12.1.el9_2       baseos          48 k
 file-libs               x86_64      5.39-12.1.el9_2       baseos         588 k
 python3-file-magic      noarch      5.39-12.1.el9_2       appstream       16 k

연결 요약
================================================================================
향상  3 꾸러미

전체 내려받기 크기: 652 k
진행할까요? [y/N]:
```

02 dnf를 이용한 패키지 설치

패키지 업그레이드

- 업그레이드할 때도 설치 과정과 똑같이 의존 관계의 패키지가 있으면 함께 업그레이드됨
- 의존성 문제를 완전히 자동으로 해결해 준다는 점이 dnf의 가장 큰 강점
- y를 입력하면 업그레이드가 진행됨

02 dnf를 이용한 패키지 설치

■ 패키지 정보 검색

- 특정 패키지의 정보를 dnf로 확인하려면 'dnf info 패키지명'을 입력함
- dnf로 패키지의 정보를 검색하면 어떤 정보가 출력되는지 다음 예를 통해 확인해 보기

```
[root@localhost 다운로드]# dnf info xterm
마지막 메타자료 만료확인(0:46:32 이전): 2023년 09월 03일 (일) 오후 03시 32분 39초.
설치된 꾸러미
이름          : xterm
버전          : 366
출시          : 8.el9
구조          : x86_64
크기          : 1.8 M
원천          : xterm-366-8.el9.src.rpm
(생략)
```

- dnf info 명령을 입력하면 패키지 이름, 버전, 출시(릴리즈), 구조(아키텍처), 크기(패키지 파일 크기), 원천(패키지 파일명) 등의 정보를 출력함

02 dnf를 이용한 패키지 설치

■ 패키지 삭제

- dnf로 패키지를 삭제하려면 'dnf remove 패키지명' 명령을 사용
- 패키지를 삭제할 때도 의존성을 확인한 후 해당 패키지와 더 이상 사용하지 않는 의존성 패키지를 함께 삭제함
- 삭제할 것인지 물어보는 질문에 y를 입력해야 패키지가 삭제됨

```
[root@localhost 다운로드]# dnf remove xterm
종속성이 해결되었습니다.
================================================================================
 꾸러미                  구조        버전                  저장소          크기
================================================================================
삭제 중:
 xterm                  x86_64      366-8.el9             @appstream      1.8 M
사용하지 않는 종속 꾸러미 제거:
 mkfontscale            x86_64      1.2.1-3.el9           @appstream      54 k
 xorg-x11-fonts-misc    noarch      7.5-33.el9            @appstream      6.1 M

연결 요약
================================================================================
삭제  3 꾸러미

사용 가능한 공간 : 7.9 M
진행할까요? [y/N]:
```

02 dnf를 이용한 패키지 설치

dnf로 패키지 설치하기

① gimp 패키지를 dnf로 설치해 보기. Gimp는 GNU Image Manipulation Program의 약자로 그래픽 편집 도구. 우선 gimp가 이미 설치되어 있는지 확인함

```
[root@localhost 다운로드]# dnf list --installed gimp
오류: 목록과 일치하는 꾸러미가 없습니다
```

02 dnf를 이용한 패키지 설치

■ dnf로 패키지 설치하기

② gimp 패키지를 설치해 보자.
다음에서 보는 것처럼 의존성이 있는 패키지를 포함하여 21개를 설치해야 한다

```
[root@localhost 다운로드]# dnf install gimp
마지막 메타자료 만료확인(0:50:38 이전): 2023년 09월 03일 (일) 오후 03시 32분 39초.
종속성이 해결되었습니다.

================================================================================
 꾸러미                  구조        버전                  저장소          크기
================================================================================
설치 중:
 gimp                    x86_64      2:2.99.8-3.el9        appstream       19 M
종속 꾸러미 설치 중:
 LibRaw                  x86_64      0.20.2-5.el9          appstream      376 k
 SDL2                    x86_64      2.26.0-1.el9          appstream      683 k
 babl                    x86_64      0.1.86-4.el9          appstream      363 k
 gegl04                  x86_64      0.4.34-2.el9          appstream      2.1 M
 gimp-libs               x86_64      2:2.99.8-3.el9        appstream      548 k
(생략)
================================================================================
설치  21 꾸러미

전체 내려받기 크기: 33 M
설치된 크기 : 162 M
진행할까요? [y/N]:
```

02 dnf를 이용한 패키지 설치

■ dnf로 패키지 설치하기

③ y를 입력하여 설치를 진행함

```
(생략)
설치되었습니다:
  LibRaw-0.20.2-5.el9.x86_64                    SDL2-2.26.0-1.el9.x86_64
  babl-0.1.86-4.el9.x86_64                      gegl04-0.4.34-2.el9.x86_64
  gimp-2:2.99.8-3.el9.x86_64                    gimp-libs-2:2.99.8-3.el9.x86_64
  imath-3.1.2-1.el9.x86_64                      jasper-libs-2.0.28-3.el9.x86_64
  libdecor-0.1.1-1.el9.x86_64                   libgfortran-11.3.1-4.3.el9.x86_64
  libmng-2.0.3-17.el9.x86_64                    libmypaint-1.6.1-9.el9.x86_64
(생략)

완료되었습니다!
```

02 dnf를 이용한 패키지 설치

■ dnf로 패키지 설치하기

④ 설치된 패키지의 상세 정보를 확인

```
[root@localhost 다운로드]# dnf info gimp
마지막 메타자료 만료확인(0:55:04 이전): 2023년 09월 03일 (일) 오후 03시 32분 39초.
설치된 꾸러미
이름          : gimp
기간          : 2
버전          : 2.99.8
출시          : 3.el9
구조          : x86_64
크기          : 92 M
원천          : gimp-2.99.8-3.el9.src.rpm
(생략)
```

02 dnf를 이용한 패키지 설치

■ dnf로 패키지 설치하기

⑤ 설치된 패키지를 실행해 보기 위해 rpm 명령으로 실행 파일을 확인

```
[root@localhost 다운로드]# rpm -ql gimp | more
(생략)
/usr/bin/gimp
(생략)
```

⑥ 실행 파일인 /usr/bin/gimp를 일반 사용자 계정에서 실행하면 gimp 창이 뜸

```
[user1@localhost ~]$ gimp &
```

02 dnf를 이용한 패키지 설치

dnf로 패키지 설치하기

그림 8-6 gimp 동작 화면

02 dnf를 이용한 패키지 설치

dnf로 패키지 설치하기

혼자해보기 dnf로 패키지 설치하기

❶ 설치할 수 있는 패키지 목록에서 gnome과 관련된 항목을 확인한다. 패키지 목록에서 gnome-photos를 선택한다(실습 상황에 따라 다른 것을 선택할 수도 있다).

❷ 의존성 때문에 같이 설치되는 패키지는 무엇인지 확인한다.

❸ 패키지를 설치한다.

❹ 실행 파일을 확인한다.

❺ 패키지를 실행시켜 다음과 같이 잘 동작하는지 확인한다.

03 파일 아카이브와 압축

03 파일 아카이브와 압축

■ 파일 아카이브

- 아카이브는 원래 '보관소, 저장소'라는 뜻이지만 리눅스에서는 '파일을 묶어서 하나로 만든 것'이라는 의미로 사용
- tar 명령은 원래 여러 파일이나 디렉터리를 묶어서 마그네틱 테이프와 같은 이동식 저장 장치에 보관하기 위해 사용하는 명령이었음
- 지금은 다른 시스템과 파일을 주고 받거나, 백업에 필요한 여러 파일이나 디렉터리를 하나의 아카이브 파일로 생성하거나, 기존 아카이브에서 파일을 추출하기 위해 사용함

03 파일 아카이브와 압축

■ 파일 아카이브

tar

- 기능 파일과 디렉터리를 묶어 하나의 아카이브 파일을 생성한다.
- 형식 tar 기능[옵션] [아카이브 파일] [파일명]
- 기능 c: 새로운 tar 파일을 생성한다.
  t: tar 파일의 내용을 출력한다.
  x: tar 파일에서 원본 파일을 추출한다.
  r: 새로운 파일을 추가한다.
  u: 수정된 파일을 업데이트한다.
- 옵션 f: 아카이브 파일이나 테이프 장치를 지정한다. 파일명을 '-'로 지정하면 tar 파일 대신 표준 입력에서 읽어들인다.
  v: 처리하고 있는 파일의 정보를 출력한다.
  h: 심볼릭 링크의 원본 파일을 포함한다.
  p: 파일 복구 시 원래의 접근 권한을 유지한다.
  j: bzip2로 압축하거나 해제한다.
  z: gzip으로 압축하거나 해제한다.
- 사용 예 tar cvf unix.tar Unix
  tar xvf unix.tar

03 파일 아카이브와 압축

■ 아카이브 생성하기: cvf 옵션

- tar 명령을 이용하여 여러 파일이나 디렉터리를 묶어서 아카이브 파일을 생성할 수 있음
- tar 명령을 실습하기 위해, 현재 사용자가 root 계정이면 일반 사용자(user1)로 돌아오고 홈 디렉터리 아래의 Test 디렉터리로 이동함

```
[root@localhost 다운로드]# exit
로그아웃
[user1@localhost ~]$ cd Test
[user1@localhost Test]$ ls
ch2/  ch3/  ch4/  ch5/  ch6/
```

- 아카이브를 생성하려면 ccreate 기능을 사용해야 함
- v 옵션은 처리 중인 파일의 이름을 화면에 출력
- f 옵션에는 생성될 아카이브의 이름을 지정하는데, f 다음에 오는 이름을 아카이브명으로 간주함

03 파일 아카이브와 압축

■ 아카이브 생성하기: cvf 옵션

- 파일명에는 특별한 제약이 없지만 확장자를 .tar로 지정하면 나중에 파일 관리를 쉽게 할 수 있어서 좋음. f가 지정되어 있으면 tar은 첫 번째 이름을 아카이브명으로, 두 번째부터 마지막 이름까지를 묶을 파일명으로 간주함

```
[user1@localhost Test]$ tar cvf ch2.tar ch2
ch2/
ch2/temp/
ch2/temp/text1
ch2/temp/text2
ch2/temp/hosts
ch2/temp/data1.cp
ch2/data1.ln
ch2/data1.sl
ch2/tmp
ch2/test
[user1@localhost Test]$ ls
ch2/  ch2.tar  ch3/  ch4/  ch5/  ch6/
```

03 파일 아카이브와 압축

■ 아카이브 생성하기: cvf 옵션

- tar 명령으로 파일을 묶어서 아카이브 파일을 만들어도 원본 파일은 그대로 있음
- 따라서 tar 파일을 만들었을 경우 원본 파일이나 디렉터리를 그대로 두면 디스크 용량을 두 배로 차지하므로 주의해야 함

03 파일 아카이브와 압축

■ 아카이브의 내용 확인하기: tvf 옵션

- 아카이브의 내용을 확인할 때는 t기능을 사용함
- v 옵션은 파일의 상세 정보를 보기 위해 지정하는 것으로, 이 옵션을 지정하지 않으면 tar 파일에 저장 된 파일의 이름만 출력됨

```
[user1@localhost Test]$ tar tvf ch2.tar
drwxr-xr-x user1/user1      0   2023-08-06   20:56   ch2/
drwxr-xr-x user1/user1      0   2023-08-06   20:56   ch2/temp/
-rw-r--r-- user1/user1    158   2023-08-06   16:51   ch2/temp/text1
-rw-r--r-- user1/user1    158   2023-08-06   16:54   ch2/temp/text2
-rw-r--r-- user1/user1    158   2023-08-06   16:54   ch2/temp/hosts
-rw-r--r-- user1/user1    158   2023-08-06   20:39   ch2/temp/data1.cp
-rw-r--r-- user1/user1    158   2023-08-06   16:47   ch2/data1.ln
lrwxrwxrwx user1/user1      0   2023-08-06   20:22   ch2/data1.sl -> data1
lrwxrwxrwx user1/user1      0   2023-08-06   20:33   ch2/tmp -> temp
-rw-r--r-- user1/user1      0   2023-12-31   12:00   ch2/test
```

- 위의 예처럼 아카이브에 포함된 파일의 상세 정보를 볼 수 있음

03 파일 아카이브와 압축

■ 아카이브의 내용 확인하기: tvf 옵션

- ls -l 명령의 결과와 비슷하며, 각 항목의 의미는 다음과 같음
  - -: 파일의 종류(b, c, d, l, s, p 등)
  - rw-r--r--: 접근 권한
  - user1/user1: 파일 소유자와 파일의 소속 그룹
  - 숫자: 파일 크기
  - 날짜와 시간: 파일을 마지막으로 수정한 일시
  - 이름: 파일명

03 파일 아카이브와 압축

■ 아카이브 풀기: xvf 옵션

- 아카이브를 풀 때는 xextract 기능을 사용함. ch2.tar을 ch8 디렉터리로 이동한 후 파일을 추출하는 예

```
[user1@localhost Test]$ mkdir ch8
[user1@localhost Test]$ mv ch2.tar ch8
[user1@localhost Test]$ cd ch8
[user1@localhost ch8]$ tar xvf ch2.tar
ch2/
ch2/temp/
ch2/temp/text1
ch2/temp/text2
ch2/temp/hosts
ch2/temp/data1.cp
ch2/data1.ln
ch2/data1.sl
ch2/tmp
ch2/test
[user1@localhost ch8]$ ls
ch2/  ch2.tar
```

03 파일 아카이브와 압축

■ 아카이브 풀기: xvf 옵션

- 파일은 아카이브를 생성할 때 저장한 원래의 경로에 복사됨
- ch2.tar은 파일을 묶을 때 경로를 현재 디렉터리로 지정했기 때문에 파일이 현재 위치에 저장됨
- 아카이브를 풀기 전에 tvf 기능을 이용하여 아카이브의 파일 경로를 확인한 후 적당한 위치로 이동하여 작업을 수행하는 것이 바람직함

03 파일 아카이브와 압축

■ 아카이브 업데이트하기: uvf 옵션

- 아카이브를 만든 다음 일부 파일 내용이 바뀌었다면 uupdate 기능을 이용하여 수정된 파일을 추가할 수 있음
- u 기능은 지정한 파일이 아카이브에 없는 파일이거나, 아카이브에 있는 파일이지만 수정된 파일일 때 아카이브의 마지막에 추가함
- 기존 파일(ch2/data)에 변경 사항이 없을 경우 u 기능을 지정하여 아카이브를 수정해도 변경되는 것이 없음
- touch 명령을 사용하여 ch2/data.ln 파일의 수정 시간을 변경한 뒤 u 기능을 사용하면 ch2/data.ln이 아카이브에 추가됨
- 이 경우 아카이브에서 파일을 추출할 때 앞에서부터 순서대로 추출되므로 나중에 추가된 파일이 마지막에 추출됨

03 파일 아카이브와 압축

■ 아카이브 업데이트하기: uvf 옵션

```
[user1@localhost ch8]$ touch ch2/data1.ln
[user1@localhost ch8]$ tar uvf ch2.tar ch2
ch2/data1.ln
[user1@localhost ch8]$ tar tvf ch2.tar
drwxr-xr-x user1/user1     0   2023-09-03   20:14   ch2/
(생략)
-rw-r--r-- user1/user1   158   2023-08-06   16:47   ch2/data1.ln
lrwxrwxrwx user1/user1     0   2023-08-06   20:22   ch2/data1.sl  → data1
lrwxrwxrwx user1/user1     0   2023-08-06   20:33   ch2/tmp       → temp
-rw-r--r-- user1/user1     0   2023-12-31   12:00   ch2/test
-rw-r--r-- user1/user1   158   2023-09-03   20:15   ch2/data1.ln  → 추가되었다.
```

03 파일 아카이브와 압축

■ 아카이브에 파일 추가하기: rvf 옵션

- 이미 생성된 아카이브에 파일을 추가하는 기능은 rreplace
- 아카이브에 파일이 이미 있는지 확인하고 추가하는 u 기능과 달리 r 기능은 지정한 파일을 무조건 아카이브의 마지막에 추가함

```
[user1@localhost ch8]$ cp /etc/services .
[user1@localhost ch8]$ tar rvf ch2.tar services
services
[user1@localhost ch8]$ tar tvf ch2.tar
drwxr-xr-x user1/user1       0   2023-09-03   20:14   ch2/
drwxr-xr-x user1/user1       0   2023-08-06   20:56   ch2/temp/
(생략)
-rw-r--r-- user1/user1     158   2023-09-03   20:15   ch2/data1.ln
-rw-r--r-- user1/user1  692252   2023-09-03   20:18   services      → 추가되었다.
```

03 파일 아카이브와 압축

■ 아카이브 생성하고 압축하기

- 아카이브를 생성하면서 동시에 압축할 수도 있음
- 압축을 하면 아카이브 파일의 크기가 작아지므로 디스크 공간이 부족할 때 이용하면 좋음
- 다음은 gzip으로 압축한 예
- gzip으로 압축할 때는 z 옵션을 사용함
- 아카이브명에 .gz를 붙여서 gzip으로 압축했음을 표시하는 것이 좋음

```
[user1@localhost ch8]$ tar czvf ch2.tar.gz ch2
ch2/
ch2/temp/
ch2/temp/text1
ch2/temp/text2
ch2/temp/hosts
ch2/temp/data1.cp
ch2/data1.ln
ch2/data1.sl
ch2/tmp
ch2/test
[user1@localhost ch8]$ ls
ch2/  ch2.tar  ch2.tar.gz  services
```

03 파일 아카이브와 압축

■ 아카이브 생성하고 압축하기

- gzip 외에 bzip2로도 압축할 수 있음. bzip2로 압축할 때는 j 옵션을 사용함
- 아카이브명에 .bz2를 붙여서 bzip2로 압축했음을 표시하는 것이 좋음

```
[user1@localhost ch8]$ tar cvjf ch2.tar.bz2 ch2
ch2/
ch2/temp/
ch2/temp/text1
ch2/temp/text2
ch2/temp/hosts
ch2/temp/data1.cp
ch2/data1.ln
ch2/data1.sl
ch2/tmp
ch2/test
[user1@localhost ch8]$ ls
ch2/  ch2.tar  ch2.tar.bz2  ch2.tar.gz  services
```

03 파일 아카이브와 압축

■ 아카이브 생성하고 압축하기

- tar 파일로 생성한 것과 압축한 파일의 크기를 비교해 보자
- 압축한 파일의 크기가 tar 파일보다 더 작음

```
[user1@localhost ch8]$ ls -l
합계 1380
drwxr-xr-x.  3   user1   user1       73   9월   3   20:14   ch2/
-rw-r--r--.  1   user1   user1   706560   9월   3   20:18   ch2.tar
-rw-r--r--.  1   user1   user1      408   9월   3   20:22   ch2.tar.bz2
-rw-r--r--.  1   user1   user1      398   9월   3   20:21   ch2.tar.gz
-rw-r--r--.  1   user1   user1   692252   9월   3   20:18   services
```

- 압축한 아카이브 파일의 내용은 tvf로 확인할 수 있으며 xvf로 추출할 수 있음

```
[user1@localhost ch8]$ tar tvf ch2.tar.gz
drwxr-xr-x user1/user1      0    2023-09-03     20:14     ch2/
drwxr-xr-x user1/user1      0    2023-08-06     20:56     ch2/temp/
-rw-r--r-- user1/user1    158    2023-08-06     16:51     ch2/temp/text1
(생략)
```

03 파일 아카이브와 압축

gzip/gunzip: .gz 파일

- gzip 명령은 지정한 파일을 압축하여 '파일명.gz' 파일을 생성

gzip

- 기능 파일을 압축한다.
- 형식 gzip [옵션] [파일명]
- 옵션 -d: 파일 압축을 해제한다.
  -l: 압축된 파일의 정보를 보여준다.
  -r: 하위 디렉터리를 탐색하여 압축한다.
  -t: 압축 파일을 검사한다.
  -v: 압축 정보를 화면에 출력한다.
  -9: 최대한 압축한다.
- 사용 예 gzip a.txt
  gzip -v b.txt c.txt

03 파일 아카이브와 압축

gzip/gunzip: .gz 파일

- gzip 명령을 사용한 예. 먼저 앞에서 실습한 ch2.tar.gz 파일을 삭제하고 압축해 보기

```
[user1@localhost ch8]$ rm ch2.tar.gz
[user1@localhost ch8]$ gzip ch2.tar
[user1@localhost ch8]$ ls
ch2/  ch2.tar.bz2  ch2.tar.gz  services
```

- gzip에서 -l 옵션을 사용하면 압축 파일에 대한 정보를 확인할 수 있음
- 파일의 압축된 크기와 원래 크기, 압축률, 원본 파일명을 출력함

```
[user1@localhost ch8]$ gzip -l ch2.tar.gz
         compressed        uncompressed  ratio  uncompressed_name
             143016              706560  79.8%  ch2.tar
```

03 파일 아카이브와 압축

■ 압축 파일의 내용 보기: zcat 명령

zcat

- 기능 gzip으로 압축된 파일의 내용을 출력한다.
- 형식 zcat [파일명]
- 사용 예 zcat abc.gz
  zcat abc

- ch2.tar.gz의 내용을 zcat로 확인하면 다음과 같음. tar로 묶여 있는 각 파일의 내용이 출력됨

```
[user1@localhost ch8]$ zcat ch2.tar.gz | more
ch2/0000755000175000017500000000000001447506523201042 5ustar  user1user1
ch2/temp/0000755000175000017500000000000001446370515501137 1 5ustar  user1user1
ch2/temp/text10000644000175000017500000000000236144636503540123620 0
ustar  user1user1127.0.0.1 localhost localhost.localdomain localhost4 localhost4.
localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
ch2/temp/text2
0000644000175000017500000000000236144636506550123670ustar  user1user1
(생략)
```

03 파일 아카이브와 압축

■ 압축 풀기: gunzip 명령

- gzip으로 압축한 파일은 gunzip 명령으로 복원할 수 있음
- gzip은 다른 압축 명령과 달리 별도의 압축 풀기 명령을 사용하지 않고 -d 옵션(gzip -d ch2.tar.gz)만 넣어 압축 파일 을 풀 수 있음

gunzip

- 기능 gzip로 압축된 파일의 압축을 푼다.
- 형식 gunzip [파일명]
- 사용 예 gunzip abc.gz
  gunzip abc

- h2.tar.gz를 gunzip으로 풀어보자

```
[user1@localhost ch8]$ gunzip ch2.tar.gz
[user1@localhost ch8]$ ls
ch2/  ch2.tar  ch2.tar.bz2  services
```

03 파일 아카이브와 압축

bzip2/bunzip2: .bz2 파일

- bzip2는 gzip에 비해 대체로 압축률은 좋지만 속도가 약간 느림
- bzip2 명령은 지정한 파일을 압축하여 '파일명.bz2' 파일을 생성함

bzip2

- 기능 파일을 압축한다.
- 형식 bzip2 [옵션] [파일명]
- 옵션 -d: 파일 압축을 해제한다.
  -l: 압축된 파일의 정보를 보여준다.
  -t: 압축 파일을 검사한다.
  -v: 압축 정보를 화면에 출력한다.
  --best: 최대한 압축한다.
- 사용 예 bzip2 abc.txt
  bzip2 -v a.txt b.txt

03 파일 아카이브와 압축

bzip2/bunzip2: .bz2 파일

- 다음은 bzip2 명령을 사용한 예
- 먼저 앞에서 실습한 ch2.tar.bz2 파일을 삭제한 다음 압축을 실행함

```
[user1@localhost ch8]$ rm ch2.tar.bz2
[user1@localhost ch8]$ bzip2 ch2.tar
[user1@localhost ch8]$ ls
ch2/  ch2.tar.bz2  services
```

03 파일 아카이브와 압축

■ 압축 파일의 내용 보기: bzcat 명령

bzcat

- 기능 압축된 파일의 내용을 출력한다.
- 형식 bzcat [파일명]
- 사용 예 bzcat abc.bz2
  bzcat abc

- 앞에서 압축한 ch2.tar.bz2의 내용을 bzcat으로 확인하면 tar로 묶여 있는 각 파일의 내용이 출력됨

03 파일 아카이브와 압축

■ 압축 풀기: bunzip2 명령

- bzip2로 압축한 파일은 bunzip2 명령으로 복원할 수 있음

bunzip2

- 기능 bzip2로 압축된 파일의 압축을 푼다.
- 형식 bunzip2 [파일명]
- 사용 예 bunzip2 abc.txt.bz2
  bunzip2 abc.txt

- 앞에서 압축한 ch2.tar.bz2를 bunzip2로 풀어보자

```
[user1@localhost ch8]$ bunzip2 ch2.tar.bz2
[user1@localhost ch8]$ ls
ch2/  ch2.tar  services
```

03 파일 아카이브와 압축

■ 아카이브 생성하고 파일 압축하기

혼자해보기 아카이브 생성하고 파일 압축하기

❶ 현재 디렉터리의 위치를 파악하고 user1의 홈 디렉터리 아래 Test 디렉터리로 이동한다.
❷ ch3 디렉터리를 아카이브 파일로 묶는다. 파일명은 ch3.tar로 한다.
❸ ch3.tar 파일의 내용을 확인한다.
❹ ch3.tar 파일을 ch8 디렉터리로 이동한다.
❺ ch3.tar 파일을 bzip2로 압축한다.
❻ ch3.tar.bz2 파일의 압축을 푼다.
❼ ch3.tar 파일을 푼다.

04 소프트웨어 컴파일

04 소프트웨어 컴파일

■ 컴파일러 설치

- 사용자가 사용하는 고급 언어(예.C 언어)를 기계어(실행 파일)로 변환하는 과정을 컴파일이라 하며, 이러한 기능이 있는 소프트웨어를 컴파일러라고 함
- 시스템에 C 컴파일러가 설치되어 있는지 알아보자.
  리눅스에서 사용하는 C 컴파일러는 GNU C 컴파일러로 패키지 이름이 gcc.
  이를 rpm 명령으로 확인해 봄

```
[user1@localhost ch8]$ rpm -qa | grep gcc
libgcc-11.3.1-4.3.el9.x86_64
```

04 소프트웨어 컴파일

■ 컴파일러 설치

- gcc 라이브러리는 설치되어 있지만 gcc 컴파일러는 없음을 알 수 있음
- dnf 로 gcc 패키지를 설치함

```
[user1@localhost ch8]$ su -
암호:
[root@localhost ~]# dnf install gcc
(생략)
설치되었습니다:
  gcc-11.3.1-4.3.el9.x86_64              glibc-devel-2.34-60.el9.x86_64
  glibc-headers-2.34-60.el9.x86_64       kernel-headers-5.14.0-284.25.1.el9_2.x86_64
  libxcrypt-devel-4.4.18-3.el9.x86_64    make-1:4.3-7.el9.x86_64

완료되었습니다!
[root@localhost ~]# exit
로그아웃
[user1@localhost ch8]$
```

04 소프트웨어 컴파일

C 프로그램 컴파일하기

- vi로 소스 코드를 바로 작성해 보자. user1의 ch8 디렉터리에서 작성함

```
[user1@localhost ch8]$ vi hello.c
#include <stdio.h>
int main() {
    printf("Hello, World.\n");
}

:wq
```

- 컴파일 명령은 간단하게 'gcc 소스 파일명'과 같이 입력.
  소스 코드에 오류가 없으면 실행 파일이 생성됨.
  실행 파일의 이름은 사용자 가 지정하지 않으면 기본적으로 a.out으로 생성됨

```
[user1@localhost ch8]$ gcc hello.c
[user1@localhost ch8]$ ls
a.out*  ch2/  ch2.tar  hello.c  services
```

04 소프트웨어 컴파일

C 프로그램 실행하기

- a.out을 실행할 수 있음
- 만약 다음과 같이 a.out을 찾을 수 없다고 나오면 현재 디렉터리가 경로에 설정되어 있지 않기 때문이므로 실행할 때 경로를 지정해야 함

```
[user1@localhost ch8]$ a.out
bash: a.out: command not found...
```

- 현재 디렉터리를 지정하여 실행하면 프로그램이 실행되어 'Hello, World.'가 출력됨

```
[user1@localhost ch8]$ ./a.out
Hello, World.
```

04 소프트웨어 컴파일

■ 실행 파일명 변경하기

- gcc로 생성한 기본 실행 파일은 a.out
- 사용자가 원하는 이름으로 지정하려면 -o 옵션을 사용

```
[user1@localhost ch8]$ gcc -o hello hello.c
[user1@localhost ch8]$ ./hello
Hello, World.
```

04 소프트웨어 컴파일

make 명령 사용하기

- gcc로 일일이 컴파일하여 하나의 실행 파일로 만드는 것은 매우 번거로운 작업인데, 이를 간단하게 해결할 수 있도록 해주는 것이 make 명령
- makefile(또는 Makefile)에 설정된 정보를 읽어서 여러 소스 파일을 컴파일하고 링크하여 최종 실행 파일을 만들어 줌

make 명령 설치하기

- make 명령이 설치되어 있는지 확인
- 만약 make 명령이 없다면 dnf install make로 설치함

```
[user1@localhost ch8]$ rpm -qa | grep make
make-4.3-7.el9.x86_64
```

04 소프트웨어 컴파일

■ 소스 파일 준비하기

- make 명령을 사용하기 위해 먼저 소스 파일을 준비
- 소스 파일은 두 개로 구성되어 있음. 각각 vi로 입력하고 저장함
- 첫 번째 파일 작성

```
[user1@localhost ch8]$ vi one.c
#include <stdio.h>
extern int two();
int main() {
    printf("Go to Module Two--\n");
    two();
    printf("End of Module One.\n");
}
:wq
```

04 소프트웨어 컴파일

■ 소스 파일 준비하기

- 이전과 같은 방식으로 입력하고 gcc로 컴파일하면 오류 메시지가 출력

two( )가 무엇인지 모르겠다는 내용

```
[user1@localhost ch8]$ gcc one.c
/usr/bin/ld: /tmp/ccSrVVGM.o: in function `main':
one.c:(.text+0x14): undefined reference to `two'
collect2: error: ld returned 1 exit status
```

- 두 번째 파일. 이 파일에 two( )가 정의되어 있음

```
[user1@localhost ch8]$ vi two.c
#include <stdio.h>
int two() {
    printf("In Module Two--\n");
   printf("--- This is a Module Two.\n");
    printf("End of Module Two.\n");
}
:wq
```

04 소프트웨어 컴파일

makefile 작성하기

- Makefile: 어떤 소스를 읽고 어떻게 컴파일하여 최종적으로 어떤 실행 파일을 만들면 되는지 make 명령에 알려주는 설정 파일
- one.c와 two.c 파일을 컴파일하고 연결하여 실행 파일 one을 만들기 위한 makefile을 간단한 형태로 만들어보면 다음과 같음

```
[user1@localhost ch8]$ vi makefile
TARGET=one
OBJECTS=one.o two.o

${TARGET} : ${OBJECTS}
        gcc -o ${TARGET} ${OBJECTS}
one.o : one.c
        gcc -c one.c
two.o : two.c
        gcc -c two.c
:wq
```

04 소프트웨어 컴파일

makefile 작성하기

- makefile에서 TARGET이나 OBJECTS는 사용자가 임의로 정한 매크로
- gcc에서 -c 옵션은 .c 파일을 컴파일하여 오브젝트 파일인 .o 파일을 생성하는데 이 .o 파일들을 연결하여 실행 파일을 만드는 것
- 실행 파일의 이름을 one으로 했으므로 다음과 같이 실행하면 됨

```
[user1@localhost ch8]$ make
gcc -c one.c
gcc -c two.c
gcc -o one one.o two.o
[user1@localhost ch8]$ ./one
Go to Module Two--
In Module Two--
--- This is a Module Two.
End of Module Two.
End of Module One.
```

Thank you!

한빛아카데미 Hanbit Academy, Inc.