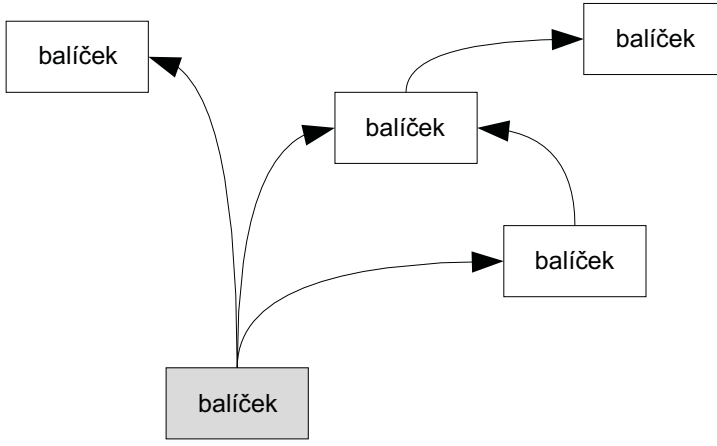
Balíčkové systémy

Balíčkové (nebo také *balíčkovací*) *systémy* určitě zná každý uživatel GNU/Linuxu, který někdy instaloval nebo aktualizoval nějaký program. Faktem je, že právě balíčkové systémy jsou jednou z hlavních výhod linuxových distribucí oproti systému Microsoft Windows, kde se většina programů instaluje a aktualizuje jednotlivě, což obvykle znamená pokaždé se „proklikat" sérií dialogů.

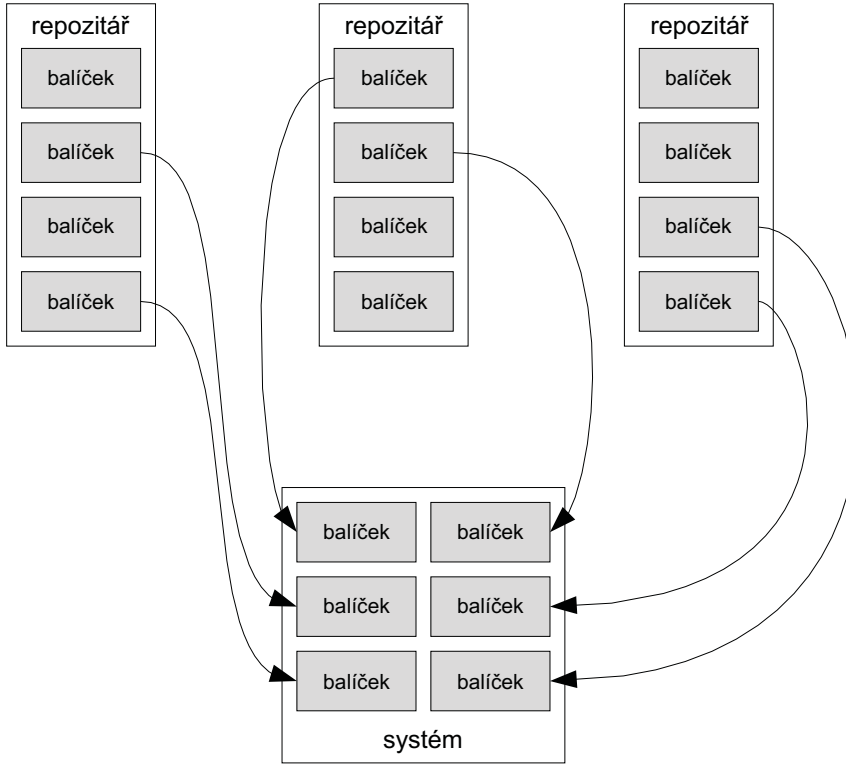
Vlastnosti balíčkových systémů

Balíčkové systémy mají několik důležitých vlastností, kvůli kterým vznikly a které jim dávají takový význam pro linuxové distribuce:

- **Jednoduchá instalace, odebírání a aktualizace programů.** V grafickém prostředí lze prakticky jediným klepnutím nainstalovat nový program nebo aktualizovat celou distribuci, v prostředí příkazové řádky je to záležitostí napsání dvou až tří slov.
- **Vyhodnocování závislostí.** Není výjimkou, že program vyžaduje řadu různých knihoven nebo jiných komponent, a to ještě v konkrétních verzích. Balíčkový systém dokáže závislosti vyhodnotit a společně s instalovaným programem nainstalovat i všechny další potřebné součásti. Na obrázku 5.1 je vidět, jak se při instalaci jednoho balíčku (označeného šedě) dostaneme přes závislosti k dalším balíčkům, které se musí rovněž nainstalovat.
- **Instalace z různých zdrojů.** Při vlastní instalaci se není potřeba zabývat tím, odkud se programy instalují. Lze instalovat z CD/DVD, většinou se ale instaluje z různých internetových repozitářů (úložišť) softwaru, poskytovaných tvůrcem distribuce nebo i někým jiným, často také ze zrcadel (mirrorů). Zdroje pro instalaci se nastaví jen jednou, příště jsou již k dispozici pro hledání a instalaci potřebných balíčků. Princip instalace balíčků z repozitářů je znázorněn na obrázku 5.2.
- **Rychlé vyhledávání.** Tím, že jsou informace o balíčcích na jednom místě, lze rychle nalézt to, co je potřeba. Místo procházení webových stránek autora programu a hledání aktuální verze stačí zadat dotaz do balíčkového systému (nebo přímo označit konkrétní program).
- **Bezpečnost dat.** Balíčky s programy lze elektronicky podepisovat, při instalaci se pak tento podpis kontroluje a lze tak snadno odhalit podvržený nebo pozměněný balíček.



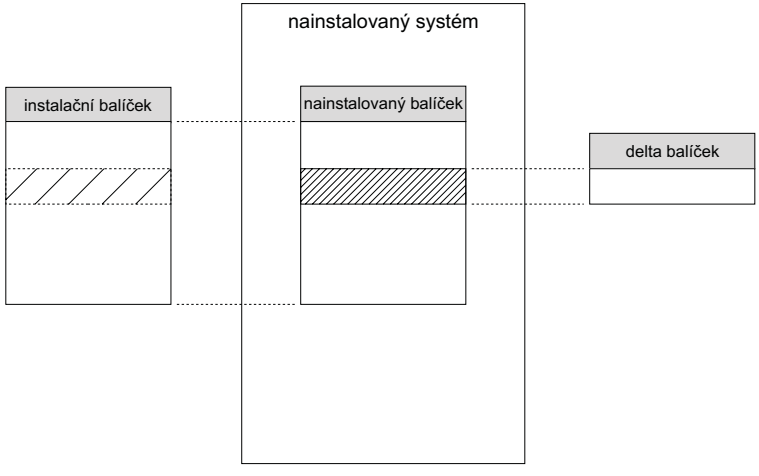
Obrázek 5.1 Závislosti mezi balíčky



Obrázek 5.2 Princip instalace balíčků z různých repozitářů

Tyto vlastnosti lze považovat za základní, i když obecně nemusí každý balíčkový systém disponovat všemi z nich (například nemusí podporovat podepisování balíčků). Pro pohodlí při práci je výhodné, když balíčkový systém podporuje ještě některé další funkce:

- **Řízení aktualizací.** Ne vždy je výhodné mít všechny programy v nejnovějších verzích. Může se například stát, že si nová verze nerozumí s hardwarem nebo že se do ní dostala jiná nepříjemná chyba. Toto lze řešit dočasným zákazem aktualizace, pokud to systém podporuje. Jiným případem aktivních zásahů do aktualizačního procesu je situace, kdy jsou stejné balíčky k dispozici ve více repozitářích a je třeba rozhodnout, zda se budou instalovat přednostně novější s více funkcemi nebo starší lépe odladěné.
- **Rozdílové balíčky** (tzv. *delta balíčky*). Řada programů je velmi objemná – i při rozdělení do samostatných balíčků podle jednotlivých komponent má balíček často desítky megabajtů. Přitom se běžně stává, že je při aktualizaci potřeba změnit jen několik souborů, které mají i o několik řádů menší velikost. Pak se hodí schopnost balíčkového systému pracovat s balíčky, které obsahují jen to, co se oproti nainstalovanému balíčku změnilo. Ocení to hlavně ti, kdo mají nepříliš rychlé internetové připojení nebo za přenášená data platí. Viz obrázek 5.3.
- **Více verzí stejné komponenty.** Nejčastěji je to potřeba u knihoven. I když by se při změně rozhraní knihovny měla změnit verze (a také označení balíčku) tak, aby nová i stará verze koexistovaly v systému bez problémů, někdy se to nepovede, případně se může stát, že program vyžaduje zcela konkrétní verzi knihovny. Proto je výhodné mít systém, který umožní existenci více verzí dané komponenty (tedy i více verzí balíčku), aniž by tyto balíčky kolidovaly.
- **Kompatibilita mezi systémy.** Někdy se stává, že balíček pro daný balíčkový systém jednoduše není k dispozici, protože ho nikdo nevytvořil – je ale k dispozici balíček jiného systému. V takovém případě se pak hodí mít možnost použít takový cizí balíček – buď přímo, nebo převedením na balíček příslušného používaného systému.



Obrázek 5.3 Princip delta balíčků

Tvoříme-li novou linuxovou distribuci, patří výběr balíčkového systému (případně tvorba nového, pokud by měla vážný důvod) mezi základní otázky, na které je potřeba nalézt odpověď. Výhodné je pokusit se o co největší kompatibilitu s již existujícími distribucemi, aby šly v případě potřeby používat balíčky z těchto distribucí, a to s co nejmenšími úpravami nebo nejlépe zcela beze změn.

Poznámka: Někdy se zcela cíleně vytvářejí distribuce, které nejsou založeny na balíčcích. Může to mít svůj význam, zejména u embedded distribucí, které se šíří v neměnné podobě a je potřeba minimalizovat jejich datovou velikost. Celá distribuce pak může mít například podobu komprimovaného archivu nebo přímo obrazu úložného zařízení (disku).

Systém RPM

Zřejmě nejpoužívanějším balíčkovým systémem v linuxových distribucích je *RPM Package Manager* (původně *Red Hat Package Manager*). Má původ v někdejších distribucích *Red Hat Linux* (z nichž se následně staly *Red Hat Enterprise Linux* jakožto enterprise distribuce a *Fedora Core*, resp. později jen *Fedora*, jakožto „distribuce pro nové technologie"). Brzy se systém RPM začal používat i v jiných distribucích, což ještě zesílilo poté, co se v něm objevily nové zajímavé funkce.

V souvislosti s balíčkovým systémem RPM je potřeba rozlišovat tři různé věci:

- systém jako takový, tedy jeho filosofii a formát balíčků,
- RPM API umožňující pracovat s balíčky, RPM databází apod. v libovolném programu,
- standardní RPM programy jako `rpm` , `rpmbuild` a další.

Z hlediska volby systému jsou nejdůležitější právě ty obecné vlastnosti, i když kvalita rozhraní a vlastnosti standardních programů mohou mít také vliv na rozhodování o tom, který balíčkový systém zvolit.

Balíčky RPM

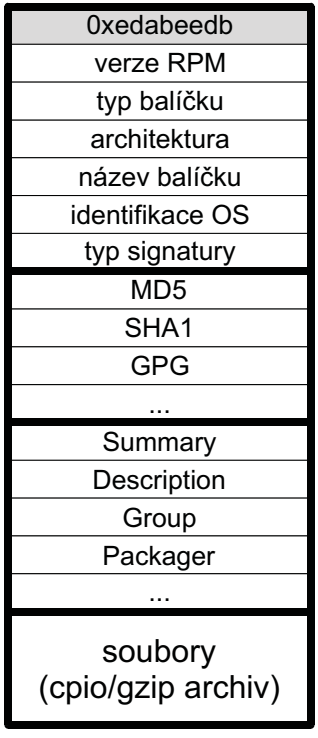
RPM balíčky mají název ve formátu `název-verze-release.architektura.rpm`. Například `mplayer-1.0-3.rc1.4.i386.rpm` je balíček programu `mplayer` ve verzi 1.0, release 3.rc1.4, pro architekturu i386. Místo architektury může být uvedeno označení `src` (balíček se zdrojovými kódy), `nosrc` (pouze soubory pro sestavení balíčku, nic dalšího) nebo `noarch` (balíček bez závislosti na architektuře – používá se například pro balíčky s dokumentací).

Samotný balíček je uvnitř složen ze čtyř sekcí: úvodu, signatury, hlavičky a archivu. Úvod obsahuje informace pro identifikaci balíčku – konkrétně magické číslo (`0xedabeedb`; podle něj lze RPM balíček rozpoznat), verzi RPM, typ balíčku (binární/zdrojový), architekturu, název, identifikaci operačního systému (pro který je balíček určen) a typ signatury. Úvod byl původně využíván systémem RPM, ale pro malou flexibilitu (např. nemožnost ukládat dlouhé názvy) se s ním v novějších verzích systému RPM již nepracuje a slouží pouze účelům vnější identifikace.

Signatura je místo, kam se ukládají informace sloužící pro kontrolu integrity (a případně autenticity) následujících dvou sekcí, tedy hlavičky a archivu, a to na základě hashů nebo podpisů (MD5, SHA1, DSA, RSA, PGP, GPG apod.).

Hlavička obsahuje mnoho důležitých informací o balíčku. Nemá příliš smysl rozepisovat, co všechno tam je uloženo – každý se tam může podívat, a to nejlépe otevřením RPM balíčku například v Midnight Commanderu (stačí na souboru s balíčkem stisknout Enter a pak otevřít k prohlížení soubor `HEADER`) nebo pomocí příkazu rpm -qi. Kromě názvu a dalších identifikačních informací je v hlavičce například stručný popis (*Summary*), podrobný popis (*Description*), informace o licenci, odkaz na zdrojový balíček (pokud je balíček binární), URL webu programu, skupina programů (kam balíček patří, například *Applications/Internet*) a různé další informace.

Poslední sekcí je již samotný archiv, kde jsou uloženy jednotlivé soubory. Tato část není v podstatě nic jiného než běžný cpio archiv, komprimovaný nejčastěji metodou gzip (ale šíří se i metoda lzma). Celou podobu balíčku RPM ukazuje obrázek 5.4.



Obrázek 5.4 Balíček RPM

Poznámka: Formát cpio je velmi starý. Byl vytvořen pro páskovou archivaci v systému PWB/UNIX a společně s utilitou cpio se stal součástí standardu POSIX.1-1988. Přestože pro archivaci zvítězil formát tar, v balíčkovém systému RPM se cpio s úspěchem používá, i když novější verze formátu RPM podporují i jiné archivační formáty.

Nástroje pro práci s balíčky RPM

Základním nástrojem je nízkoúrovňový program `rpm`. Je určen pro jednoduché činnosti – tedy instalaci nebo odinstalaci balíčku, aktualizaci nainstalovaného balíčku, kontrolu integrity balíčku (kontrola hashe nebo podpisu), dotazování na vlasnosti balíčku (např. na závislosti nebo na to, co balíček poskytuje), podepisování balíčků a různé pomocné operace (inicializace databáze balíčků, import veřejných klíčů apod.).

Program `rpm` postačuje na uvedené jednoduché činnosti, pro rutinní správu systému se však nehodí. Hlavními nevýhodami je totiž absence důležitých funkcí, včetně automatického vyhodnocování závislostí a získávání balíčků z definovaných úložišť.

Uvedené nevýhody nemají programy „vyšší úrovně". Většinou fungují jako front-end k programu `rpm`. Mezi takové programy patří zejména `yum` (používá ho např. Red Hat Enterprise Linux, Fedora, Yellow Dog Linux a další), `zypper` (openSUSE, SUSE Linux Enterprise), urpm (Mandriva), `apt-rpm` (součást programu APT používaného na distribuci Debian a dalších), *Synaptic Package Manager* (distribuce Ubuntu ad.) a některé další aplikace. Teprve s programy tohoto druhu získává balíčkový systém svou plnou sílu, protože veškeré operace s balíčky nebo i celým systémem fungují prakticky „na jedno kliknutí", resp. na několik stisků kláves.

Tip: S praktickým použitím některých z uvedených nástrojů (a s dalšími uvedenými u dalších balíčkových systémů) se můžete seznámit ve druhé části knihy (4. kapitola).

Systém Debian

Balíčkový systém distribuce Debian (označovaný i jako „deb" podle přípony názvu souborů s balíčky) je srovnatelně vyspělý jako RPM. Liší se ovšem formát balíčků a způsob práce s nimi.

Tento balíčkovací systém se používá kromě v Debianu samozřejmě i v Ubuntu, jinak však není (alespoň co do počtu distribucí) tak rozšířený jako RPM. Za zmínku stojí třeba jeho použití v embedded zařízeních v rámci metadistribuce OpenEmbedded.

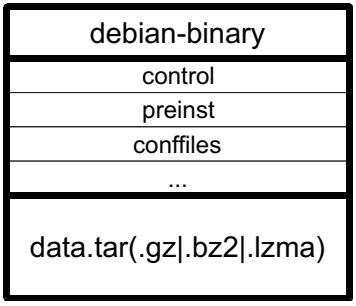
Balíčky Debian

Balíček systému Debian je vlastně archiv typu `ar` (stejný, jako používají například statické knihovny v GNU/Linuxu). Obsahem tohoto archivu jsou typicky tři soubory: `debian-binary` (obsahuje číslo verze formátu, aktuálně 2.0), `control.tar.gz` (metadata balíčku) a `data.tar` (vlastní data balíčku).

Poznámka: Soubor `data.tar` může být – a obvykle bývá – komprimován pomocí GZIP, BZIP2 nebo LZMA. Pak má název podle příslušné komprese, tedy `data.tar.gz`, `data.tar.bz2`, resp. `data.tar.lzma`.

Název balíčku má tvar `název_verze-debianverze_architektura.deb`. Jednotlivé složky mají podobný význam jako u balíčků RPM. Například soubor `postfix_2.5.5-1.1_i386.deb` je balíček s programem *Postfix* ve verzi 2.5.5, verze Debian je 1.1 a architektura i386.

Obsahem archivu `control.tar.gz` jsou soubory s metadaty a skripty související s instalací/odinstalací balíčku. Například soubor `control` obsahuje informace o verzi, závislostech, konfliktech, navrhovaných balíčcích atd., skripty `preinst` a `postinst` se spouštějí před, resp. po instalaci, soubor `conffiles` obsahuje seznam všech konfiguračních souborů v balíčku, v souboru `md5sums` jsou MD5 hashe souborů apod.



Obrázek 5.5 Struktura balíčku Debian

Poznámka: Zajímavou informací (uloženou v souboru `control`) je priorita balíčku. Říká, do jaké míry je balíček důležitý pro systém. Priorita `required` znamená balíček bezpodmínečně nutný pro funkci systému (pokud je ještě navíc označen jako `essential`, program pro správu balíčků ho odmítne odinstalovat), `important` jsou označeny důležité systémové balíčky, `standard` jsou balíčky instalované ve výchozím nastavení, `optional` nepovinné balíčky a konečně `extra` jsou balíčky, které jsou nepovinné a navíc mohou například kolidovat s balíčky s vyšší prioritou.

Zdrojové „balíčky" Debian vlastně žádnými balíčky nejsou. Je to sada samostatných souborů, která obsahuje původní zdrojový archiv (tzv. *upstream*, od autora), patch se změnami pro balíček (veškeré řídicí soubory balíčku a změny provedené na souborech z původního archivu) a definiční soubor balíčku (`*.dsc`). Definiční soubor obsahuje důležité informace o balíčku a může být považován za zdrojový balíček jako takový, byť jsou samozřejmě nezbytně potřeba i zmíněné dva soubory. Podrobně se tím zabývá 4. kapitola druhé části knihy.

Nástroje pro práci s balíčky Debian

Podobně, jako u systému RPM existuje program `rpm`, u balíčkového systému Debian je podobný nízkoúrovňový nástroj `dpkg`. Téměř stejná je také množina činností, které je `dpkg` schopen provádět.

Pro pohodlnější práci je opět potřeba sáhnout po něčem jiném. Základním nástrojem je *Advanced Packaging Tool* (*APT*), již zmíněný u systému RPM a pracující na základě parametrů zadávaných z příkazové řádky. Ještě o něco komfortnější jsou další programy: *Aptitude*, *Synaptic*, *KPackage* a různé jiné.

Archivy .tgz

Tento druh „balíčků" vlastně žádné balíčky nejsou. Vypadají jako archivy se soubory (archiv `tar` komprimovaný metodou GZIP), nicméně některé linuxové distribuce (zejména Slackware) je používají jako formát pro balíčkový systém.

S balíčky založenými na `tgz` archivech se pracuje poměrně jednoduše a spolehlivě, postrádají však jednu poměrně zásadní věc: řešení závislostí. To je důvod, proč se takový systém hodí jen pro distribuce určené zkušeným administrátorům, kteří dokáží takové „drobnosti", jakou je nalezení a instalace všech potřebných komponent, hravě vyřešit. Běžný uživatel z takto fungujícího systému radost mít nebude.

Poznámka: Co je pro většinu případů nevýhodou, může být někdy i výhodou. Mnohé programy – aby nedocházelo k rozdrobování funkčních celků do většího počtu balíčků – mívají závislosti, které ve skutečnosti vůbec nepotřebují. Ignorovat závislosti lze i u systémů, jako je RPM, ale pokud administrátor není zvyklý tyto věci řešit, raději se smíří s nabobtnalým systémem, než aby se do spleti závislostí ponořil a riskoval případné problémy.

Archivy se zdrojovými kódy

Jednou z možností, jak řešit práci s balíčky, je založit celý systém na kompilaci ze zdrojových kódů. Idea je založena na tom, že pokud je archiv se zdrojovými kódy připraven standardními nástroji ze sady *autotools* (a pro kompilaci a instalaci lze použít tzv. „svatou trojici", tedy `./configure`, `make` a `make install`), měl by jít program vždy nainstalovat – samozřejmě jsou-li splněny všechny požadavky na prostředí.

Takovým způsobem funguje například distribuce *Gentoo*, která však potřebuje ještě balíčky s dalším informacemi (metadaty) – viz *Portage*. Drtivá většina systému se při instalaci kompiluje ze zdrojových kódů, taktéž každá aktualizace znamená další kompilaci. Má to své výhody:

- Program se kompiluje pro konkrétní prostředí, bez nutnosti dělat kompromisy.
- Lze si přesně zvolit, co bude součástí programu (co se bude kompilovat).
- Lze stlačit vyžadované závislosti na minimum.
- Aktualizace systému lze provádět ihned po vydání zdrojových kódů, není třeba čekat na binární balíky.

Nelze ovšem přehlédnout ani nevýhody:

- Vyhodnocení závislostí se musí provádět ručně nebo externím mechanizmem.
- Konfigurace kompilací může být velmi pracná a zdlouhavá.
- Kompilace může trvat dlouho (zejména na pomalejších strojích), řádově hodiny až dny.

Poznámka: Odpůrci distribucí založených na kompilaci ze zdrojových kódů občas přidávají ještě jeden protiargument: tyto distribuce, přehnaně řečeno, přispívají ke globálnímu oteplování a při nasazení v masovém měřítku by mohly destabilizovat přenosovou elektrickou soustavu. Pravdou je, že kompilace patří k procesorově náročnějším činnostem.

Ostatní balíčkové systémy

Poněkud stranou zájmu existují ještě další balíčkové systémy, používané většinou jen u specifických distribucí. Přesto si dva z nich zaslouží alespoň pár slov, protože disponují zajímavými vlastnostmi.

PiSi

PiSi (též *Packages Installed Successfully as Intended*) je balíčkový systém použitý u linuxové distribuce *Pardus*. Má jediný nástroj napsaný v jazyce Python, který poskytuje nízkoúrovňové i vysokoúrovňové operace, jak je známe z jiných systémů. Databáze balíčků je uložena v *Berkeley DB*. Pro kompresi dat v balíčcích se používá kompresní algoritmus *LZMA*.

Portage

Portage je balíčkový systém využívaný distribucí *Gentoo*. Typicky instaluje ze zdrojových kódů (pracuje s balíčky `ebuild`, z nich získává potřebné informace, pak stahuje a kompiluje zdrojové kódy). Výhodou tohoto systému je velká flexibilita – cílový systém se tvoří na základě předem definovaných kritérií (hlavně definice v proměnné `USE` a definice platformy/architektury). Lze ovšem instalovat i binární (předkompilované) balíčky.

Balíčky `ebuild` jsou určitý způsob uložení metadat o zdrojových kódech, z nichž se instalují jednotlivé součásti distribuce. Každý `ebuild` je vlastně shellový skript obsahující jak výkonný kód (příkazy pro instalaci a další operace), tak popis balíčku, licence, informace o tom, kde se nacházejí zdrojové kódy atd.

Tip: Chcete-li vytvořit vlastní distribuci bez uvažování o konstrukci systému, je právě Gentoo a jeho systém Portage zajímavou volbou. Pomocí definic lze poměrně zásadně určit podobu cílového systému a optimalizovat ho pro konkrétní platformu a konkrétní účel. Výhodou a současně nevýhodou je, že mnohé součásti (zejména různé skripty, výchozí konfigurace apod.) jsou připraveny předem (typicky již od autora příslušného programu) a není potřeba se jimi nějak zvlášť zabývat.