Unixové systémy

Petr Osička

KATEDRA INFORMATIKY
UNIVERZITA PALACKÉHO V OLOMOUCI

O KURZU

Obsah

- Základní principy unixových systémů. Práce v nich na úrovni pokročilého uživatele, práce v shellu, základní zpracování textu.

Zkouška

- Vyřešení 2 jednoduchých úkolů, u počítače, veškerá dokumentace dostupná.

Literatura:

- Vilém Vychodil, *Linux: Příručka českého uživatele*, Computer Press, Brno 2003, ISBN 80–7226–333–1.
- kolektiv: *Linux: Dokumentační projekt, 4. aktualizované vydání*. Computer Press, 2008. ISBN 978–80–251–1525–1.
- Mark G. Sobell. *A practical guide to Linux. Commands, Editors, and Shell programming*. Prentice Hall, 2013 (3. edice).
- internet (manuály), viz odkazy na webu předmětu.

Operační systém

- *jádro*
  základní softwarové vybavení počítače, které se stará o správu systémových zdrojů (např. hardware, ovladače, procesy, pamět, filesystém). Uživatelským programům poskytuje API.
- *základní programy*
  uživatelská správa hardware, dat a procesů, ovládání počítače, základní uživatelské programy

Předchozí označováno jako *base systém*.

Termín OS někdy zahrnuje i další uživatelské programy (mailový klient, web browser, kancelářský balík, multimediální programy), které jsou nainstalovány současně s base systémem.

Unixový operační systém

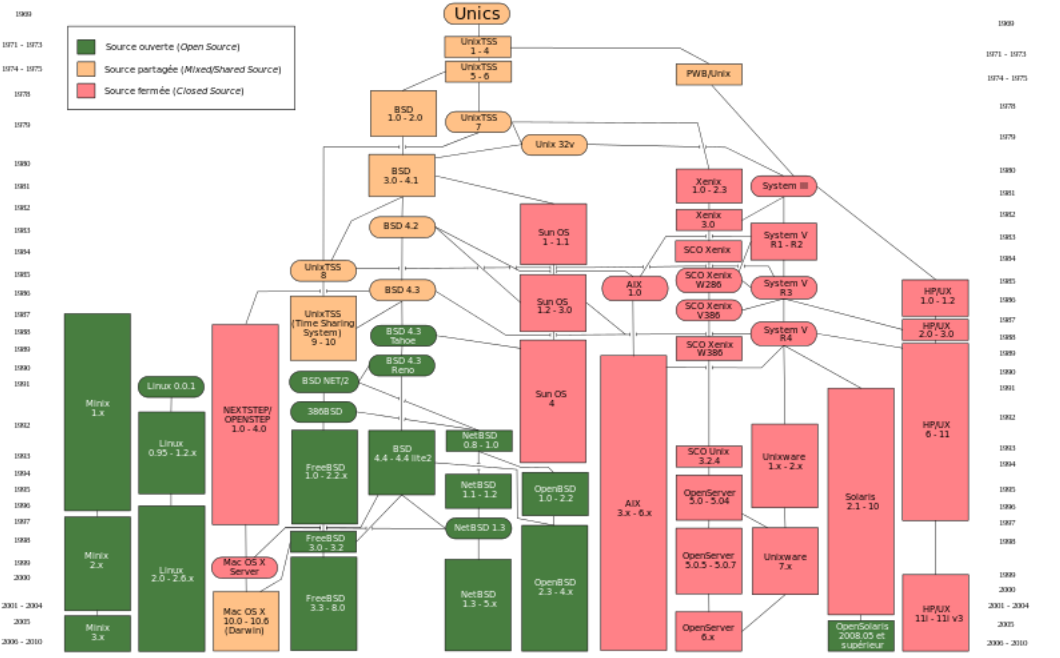
- systém koncepčně a technicky vycházející z (původního) unixu
- příklady: linuxové distribuce, BSD rodina, macOS rodina, android
- velmi rozšířené (klastry, servery, průmysl, mobily, elektronika …)

Unixová filozofie, vybrané význačné vlastnosti

- příkazový interpret (shell)
- jeden program dělá jednu věc, ale pořádně
- programy jdou skládat dohromady: výstup jednoho programu je vstupem jiného programu (převážně textový formát)
- všechno je soubor: například zařízení jsou v systému reprezentována soubory

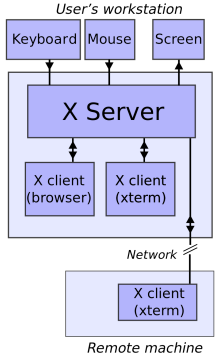
Vznik Unixu a jeho vývoj

- 60. léta. Bellovy laboratoře: CTSS, Multics
- konec 60. let již UNIX, DEC PDP 7 (velká část napsána K. Thomsonem za 3. týdny) poté DEC PDP 11
- zlom je vznik programovacího jazyka C a přepsání unixu do C. To spolu s přenositelným překladačem jazyka C zajistilo přenositelnost unixu.
- 6. verze (cca 9000 řádků kódu), *Lion's commentary on Unix 6th edition*
- od té doby bouřlivý vývoj v komerčních, akademických i svobodných/open-source verzích (unix wars)
- IEEE POSIX standard (systémové rozhraní atd).
- Od 80. let projekt GNU, FSF, R. M. Stallman (GPL licence a copyleft ...)
- 1983 — Turingova cena pro K. Thomsona a D. Ritchieho za Unix/C
- Od 1991, Linus Torvalds, linuxové jádro, později linuxové distribuce s GNU nástroji


Unics
Source ouverte (Open Source)
Source partagée (Mixed/Shared Source)
Source fermée (Closed Source)
1969
1971 - 1973
1974 - 1975
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001 - 2004
2005
2006 - 2010
UnixTSS 1 - 4
UnixTSS 5 - 6
UnixTSS 7
PWB/Unix
BSD 1.0 - 2.0
Unix 32v
BSD 3.0 - 4.1
Xenix 1.0 - 2.3
System III
Xenix 3.0
BSD 4.2
Sun OS 1 - 1.1
System V R1 - R2
SCO Xenix
UnixTSS 8
BSD 4.3
Sun OS 1.2 - 3.0
AIX 1.0
SCO Xenix V/280
System V R3
HP/UX 1.0 - 1.2
SCO Xenix V/386
HP/UX 2.0 - 3.0
UnixTSS (Time Sharing System) 9 - 10
BSD 4.3 Tahoe
System V R4
BSD 4.3 Reno
SCO Xenix V/386
BSD NET/2
Sun OS 4
386BSD
Minix 1.x
Linux 0.0.1
NEXTSTEP/ OPENSTEP 1.0 - 4.0
NetBSD 0.8 - 1.0
SCO Unix 3.2.4
Unixware 1.x - 2.x
HP/UX 6 - 11
FreeBSD 1.0 - 2.2.x
BSD 4.4 - 4.4 lite2
NetBSD 1.1 - 1.2
OpenBSD 1.0 - 2.2
OpenServer 5.0 - 5.04
Solaris 2.1 - 10
Linux 0.95 - 1.2.x
AIX 3.x - 6.x
Minix 2.x
NetBSD 1.3
FreeBSD 3.0 - 3.2
OpenBSD 2.3 - 4.x
OpenServer 5.0.5 - 5.0.7
Unixware 7.x
Linux 2.0 - 2.6.x
Mac OS X Server
HP/UX 11i - 11i v3
Minix 3.x
Mac OS X 10.0 - 10.6 (Darwin)
FreeBSD 3.3 - 8.0
NetBSD 1.3 - 5.x
OpenServer 6.x
OpenSolaris 2008.05 et supérieur

Grafická rozhraní

- X server (protokol a implementace), Wayland
- Desktop Environment vs okenní manažer
- nejznámější velké DE: KDE, GNOME, XFCE
- obrovské možnosti konfigurace (pro vlastní workflow)
- inspirace z jiných systémů, inovace



Linuxové distribuce

- linuxové jádro a GNU utility
- balíčky, balíčkovací systém, repozitáře
- vybrané a předkonfigurované DE, vybrané programy (např. webový prohlížeč, terminál, etc)
- provedená technická rozhodnutí: filesystem, start systému, kde je konfigurace
- updatování: rolling release, nová verze každých x měsíců
- cílová skupina
- příklady: Fedora, Debian, Arch, Gentoo, Manjaro, Mint, Ubuntu, OpenSuse …

Základy shellu

příkazová řádka, interpretuje a provádí příkazy, textové rozhraní

`bash` (rozšířený, budeme jej používat), další: `zsh`, `fish`, `kornshell`, …

Příklady příkazů

```
whoami,
pwd,
hostname,
ls,
echo,
uname,
arch,
who,
exit
```

Obecně

```
cmd [options] [parameters]
```

Typ

- vestavěný příkaz shellu (*builtin*), manuál: `help`
- samostatný spustitelný program, manuál: `man, info`

Zjišťování toho, kde hledat informace

```
type          # zjisti typ prikazu
apropos       # prohledava jmena a popisy manualovych stranek
whatis        # najde manualova stranky odpovidajici argumentu
which         # najde umisteni binarniho souboru prikazu
whereis       # najde umisteni binarniho souboru a dokumentace prikazu
```

Ovládání:

- doplňování: tab
- skok na začátek a konec: ctrl-a, ctrl-e
- vyjmutí & vložení řádku: ctrl-k, ctrl-u, ctrl-y
- existuje mnoho dalších klávesových zkratek, stačí vyhledat na internetu slovní spojení *bash keyboard shortcuts*

konfigurace (`.bashrc` – non-login shell, `.bash_profile` – login shell)

Prompt a jeho konfigurace

- proměnná `PS1`
-

`\d`	Date	`\h`	Host
`\n`	Newline	`\t`	Time
`\u`	Username	`\W`	Current working directory
`\w`	Full path to current directory		

- `PS1='\n\W\n[\h][\u]->'`

Historie příkazů

```
history
-c          vyprazni historii
-d #        smaze #-tou polozku z historie
-w          zapisou soucasnou historii do souboru
-r          nacte historii ze souboru

:parametr
#           zobrazi poslednich # polozek v historii


:v bashi

!#                #-ta instrukce v historii
!!                posledni instrukce v historii
!string           posledni istrukce, ktera zacina string
```

procházení pomocí šipek (ctrl-p, ctrl-n), prohledávání historie (ctrl-r): enter provedení nalezeného příkazu, (ctrl-g) ukončení prohledávání.

Proměnné prostředí

bez typu, bere se jako posloupnost znaků

Tisk proměnné

```
echo $var
```

Zajímavé (systémové proměnné) proměnné

```
PS1           # prompt prvni urovne
HISTSIZE      # velikost historie
HOME          # domovsky adresar aktualniho uzivatele
PWD           # aktualni adresar
OLDPWD        # predchozi aktualni adresar
SHELL         # defaultni shell
PATH          # adresare, kde shell vyhledava spustitelne soubory
```

Alias

```
alias name=command
```

delší příkaz mezi `' '`

- méně psaní
- kratší jména pro komplikované instrukce
- bez překlepů (`alias sl=ls`)
- bezpečné příkazy (`alias rm='rm -i'`)
- verze bez alias (`\prikaz`)

```
unalias       # zrusi alias
alias         # vypise aktualni alias
```

Přesměrování vstupu a výstupu

`stdin, stdout, stderr`

file descriptory: 0, 1, 2

- `>, >>`
  stdout do souboru, existující soubor je přepsán (>) nebo připojení na konec (>>)
- `|`
  použij výstup jednoho programu jako vstup do jiného programu, *roura, angl. pipe*
- `<, <<`
  přesměrování stdin, ukončení zadávání z stdin
- pro přesměrování `stderr` použijeme `2>`, např. `cmd 2> /dev/null` přesměruje chybový výstup `cmd` do soubouru `/dev/null`.
- přesměrování `stderr` na `stdout`: `2>&1`
- přesměrování `stderr` a `stdout` do stejného souboru: `&> file`
- presměrování `stdout` a `stderr` do roury: `|&`

Příklady

```
echo $PATH > foo.txt
echo dalsi radek >> foo.txt

echo foo.txt > files.txt
cat < files.txt

cat << done
cat << done > fruit

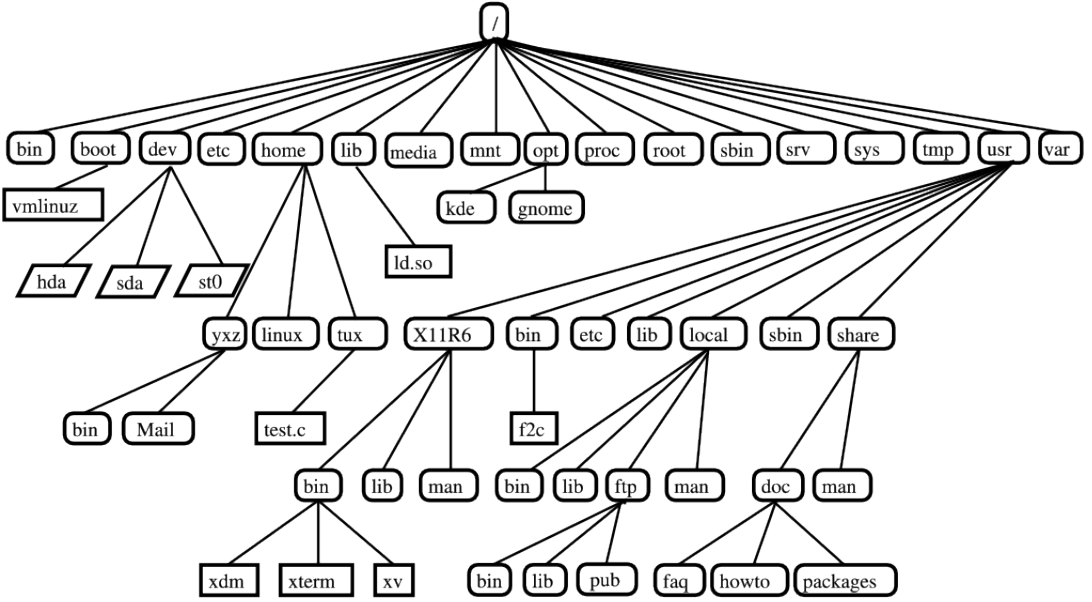
ls -1 | wc -l
```

Souborový systém

- logický způsob uspořádání uložení dat na (částech) ukládacích zařízení do jedné stromové struktury, abstrakce
- podpora mnoha FS: EXT[2,3,4], (V)FAT, NTFS, ZFS, Btrfs atd...
- připojování a odpojování pod adresář v hierarchii
- soubor: atomický kus uložených dat (z logického pohledu)
- adresář: kolekce souborů a jiných adresářů (lze chápat jako speciální soubor obsahující tuto kolekci).
- cesty: relativní (cesta stromem z nějakého místa), absolutní (cesta stromem od kořene), jednotlivé uzly odděleny lomítkem, jména oddělena znakem `/`, aktuální adresář `.`, nadřazený adresář `..`

```
/usr/share/local/bin/

relativni cesta z adresare /home/tux do adresare /home/yxz/bin
../yxz/bin
```


/
bin
boot
dev
etc
home
lib
media
mnt
opt
proc
root
sbin
srv
sys
tmp
usr
var
vmlinuz
hda
sda
st0
kde
gnome
ld.so
yxz
linux
tux
X11R6
bin
etc
lib
local
sbin
share
bin
Mail
test.c
f2c
bin
lib
man
bin
lib
ftp
man
doc
man
xdm
xterm
xv
bin
lib
pub
faq
howto
packages

Inode, linky

- *inode*
  struktura reprezentující soubor, obsahuje metainformace a ukazatel na data každý inode má číslo, (pro adresář obsahuje inode čísla obsahu adresáře).
- *hardlink*
  jméno a inode číslo,
- *symlink, symbolický link*
  soubor obsahující absolutní cestu, (i pro adresáře)

V hierarchii jsou vlastně hardlinky a symlinky. Inode pro soubor může mít více hardlinků, má-li 0 hardlinků, je odstraněn. Inode pro adresář má může mít pouze jeden hardlink (acykličnost grafu souborů). Pokud neexistuje objekt, na který odkazuje cesta v symlinku, není soft-link smazán (ale označujeme ho jako zlomený).

```
ls -i        # zobrazi inode cisla polozek
```

Globbing

Shell provádí expanzi (textu zapsaného do příkazové řádky), částí toho je nahrazování speciálních řetězců a konstrukcí. Lze využít pro vytvoření (skutečných cest).

`*`	0 nebo více libovolných znaků
`?`	jeden libovolný znak, podle obsahu FS
`[chars]`	jeden ze znaků v řetezci chars
`[c1-c2]`	jeden ze znaků v rozsahu `c1-c2`, např. `[0-9]`, `[a-z]`
`{word1,word2,...}`	jedno ze slov, nekontroluje s FS!!
`[!chars]`	jakýkoliv znak, který není v chars
`~`	zkratka domovského adresáře

```
echo *.txt        -> vsechny soubory s koncovkou txt soubory v aktualni adresari
echo [abc]*       -> vsechny soubory zacinajici jednim z pismen abc
echo /home/osicka/{doc,bin} -> /home/osicka/doc /home/osicka/bin
```

Expanze je potlačena uvnitř uvozovek a apostrofů.

Základní příkazy

`cd`	změna aktuálního adresáře
`mv`	přejmenování, přesun
`cp`	kopie
`rm`	smazání
`mkdir, rmdir`	vytvoření, smazání prázdného adresáře
`ln`	vytváření linků (hardlinků i symlinků)
`du`	velikosti adresářů a souborů
`ls`	výpis obsahu adresáře
`cat`	spojení souborů na stdin
`less`	zobrazení souboru v obrazovkovém režimu
`file`	zjištění typu souboru
`wc`	počty znaků, slov, řádků v souboru

Hledání souborů

```
find [dir] [options]

[dir]     -> kde hledáme
[options] - klasické options (napr. co delat se symlinky, hloubka)
          - specifikace vyhledacich podminek
          - zpracovani vysledku
```

Vyhledávací podmínky (příklady)

```
find /etc/ -name "*.conf"             shoda ve jmene (lze pouzit wildcards)
find ~ -size +400M                    soubory vetsi nez 400 MB
find ~ -size +400M -a -name "*.mkv"   kombinace podminek (lze -a, -o, !)
                                      pro prioritu zavorky
```

Co dělat s vysledky (výběr)

```
-delete
-exec command              pro kazdou shodu spusti command
-ok   command              jako exec, pta se na potvrzeni
-print format              formatovany vystup
```

```
locate [options] string
```

- hledá podle shody se jménem
- používá databázi (kde se indexuje obsah filesystému) tu je potřeba vytvořit a udržovat (`updatedb`)

Klasická unixová práva

- cíl: ochrana před zneužitím systémových prostředků
- pro soubory a adresáře: kdo přistupuje vs co dělá
- *kdo přistupuje*
  vlastník (`u`), skupina (`g`), zbytek (`o`)
- *co dělá*
  čtení (`r`), zápis (`w`), spuštění (`x`)
- Pro každého 8 typů přístupu, podmnožiny `{r,w,x}`.
- Lze prohlížet pomocí `ls -l` (v pořadí `ugr`, `rwx`).

	soubor	adresář
r	prohlížení, kopie	vidí obsah, např. `ls`
w	přepsání	může přidávat soubory
x	může spustit	může do něj vstoupit, např. `cd`

Změna práv

```
chmod permissions files-and-dirs
```

Dva základní způsoby zadání `permissions`

1. kdo: `a,u,g,o` (a znamená pro všechny), co: `r,w,x`, update: `+,-,=`

```
chmod g-w,o-r file.txt            # skupine odebereme zapis, ostatnim cteni
chmod u=rwx,g=r,o=  file.txt      # vlastnik vse, skupina zapis, ostatni nic
```

   `+,-,=` lze kombinovat v jednom příkazu

2. `r -> 4 (100), w -> 2 (010), x -> 1 (001)`. Práva vyjadříme číslicí, která vznikne součtem (bitovým součtem), např. `{r,x}` je `5 (101)`. Argument parameters jsou tři číslice, v pořadí `ugr`.

```
  chmod 750 file.txt      # vlastnik vse, skupina cteni,spusteni, ostatni nic
```

Sticky bit

- pro adresáře, do kterého je povolen zápis
- o může do adresáře zapisovat. Editovat soubory (včetně mazání) může pouze jejich vlastník.

- ```
  chmod o+t dir           # nastaveni sticky bit 1. zpusobem
  chmod 1722 dir          # nastaveni sticky bit 2. zpusobem
  ```

Group ID bit: u správy procesů

Další způsoby, např. SELINUX
  ```

Významné adresáře

`/bin`	uživatelské spustitelné soubory
`/etc`	konfigurační soubory
`/home`	domovské adresáře uživatelů
`/lib`	sdílené knihovny
`/root`	domovský adresář správce
`/sbin`	systémové spustitelné soubory
`/usr`	aplikace atd
`/var`	cache, mail, log, spool
`/dev`	zařízení, `/dev/null`, `/dev/zero`

Procesy

Proces je aktivní entita zhruba odpovídající běžícímu programu

- má přiděleny zdroje: pamět, otevřené soubory, síťová spojení,
- OS udržuje jeho stav: např. PC (program counter), SP (stack pointer) a rozhoduje o jeho plánování (= přidělování procesorového času) a přidělování jiných zdrojů.

Správa

- `pid` - jednoznačný identifikátor procesu (zkratka *process id*)
- proces spouští uživatel (technicky ale jiný proces)
  1. kopie spouštěcího procesu (`fork`)
  2. spustí se program, který nahradí kopii (`exec`)
- mezi procesy je tedy vztah rodič a potomek
- existuje proces, který se spouští jako první a je předkem všech ostatních: typicky `init`, má `pid` rovno 1.

Přístupová práva procesů

- proces má vlastníka a skupinu, od kterého dědí přístupová práva. Obvykle uživatel, který jej spustil
- speciální bit v přistupových právech spustitelného souboru: je-li nastaven, je vlastníkem procesu vlastník spustitelného souboru, nikoliv uživatel, který jej spustil: to umožňuje přístup takového procesu do filesystému tam, kam uživatel nemá přístup (některé systémové procesy mají vlastní účty, např. `lpd` pro tisk).
  Lze nastavit pomocí `chmod`, kde se `x` pro `u` nebo `g` nastaví na `s`.

Monitorování procesů

```
ps
# prepinace (chaos)
# -o pid,ppid,command ...
# a (bsd styl, vsichni uzivatele)
# x (process nemusi mit navazan terminal)
# -U user1,user2 ... (uzivatel, ktery vytvoril proces)
# -u user1,user2 ... (uzivatel, jeho prava proces ma)
```

- bez argumentů pouze potomci aktuálního shellu

```
top (napoveda pod znakem ?)
htop
pstree
```

Priorita procesů (jak je důležité mu přidělovat prostředky OS)

- číslo v rozsahu -20 až 19, čím menší tím je priorita vyšší
- standardně s prioritou 0, lze změnit pomocí `nice`, `renice`. Obyčejní uživatelé mají jenom kladné priority, mohou měnit jenom na vyšší číslo.

Spouštění procesů ze shellu

- jménem spustitelného souboru, pokud je umístěn někdy v adresáři v `PATH`.
- cestou (relativní nebo absolutní) ke spustitelnému souboru (např. `./filename`)
- procesy na pozadí (bez interakce s uživatelem, shellu zůstane standardní vstup, standardní výstup je sdílený) a na popředí (proces má standardní vstup i výstup), výpis job number a pid.
- spuštění na pozadí: `&` za příkazem; může být nutné přesměrovat výstup
- výpis procesů (spuštěných z aktuálního terminálu)

```
jobs
```

- přesun z popředí do pozadí/na popředí:

```
ctrl-z         # pozastaveni
bg job-number # rozbehnuti na pozadi
fg job-number # presun do popredi
```

když vynecháno `job-number`, bereme poslední úlohou (na popředí, spuštěnou).

Ukončení procesu

Uživatelské ukončení

```
kill -SIGNAL pids
```

- obecný mechanismus posílání signalů procesům, některé proces zpracuje podle svého, některé nemůže ignorovat
- signál č. 9 (`SIGKILL`): okamžité ukončení procesu, nelze ignorovat
- proces běžící v shellu na popředí lze ukončit ctrl+c, signál `SIGTERM`, proces jej může obsloužit jinak než ukončením.

Po konci procesu

- process vrací rodiči návratovou hodnotu (0 ok, jinak kód chyby)
- jsou ukončeni i potomci, pokud to není zařízeno jinak
- některé procesy to zařídí sami (např. shell, změna rodiče)
- `nohup`: proces běží i po odhlášení uživatele

Práce s textem

Regulární výraz (regex)

Fixujeme abecedu znaků Σ neobsahující rezervované znaky (viz dále).

Slovo (řetězec) nad Σ je konečná posloupnost znaků ze Σ. Jazyk nad Σ je množina slov nad Σ. Pro slova u a v pomocí uv značíme slovo vzniklé jejich zřetězením (zapojením za sebe). Prázdné slovo značíme ϵ.

Např. pro jazyk $\Sigma = \{\texttt{a, b, c}\}$ je `aaabc` slovo. Množina {abc, aa, aaab, abc} nebo množina všech slov začínajících znakem a jsou jazyky.

Regulární výraz (regex) je slovo nad abecedou obsahující Σ obohacenou o rezervované znaky, vytvořené podle pravidel níže, které slouží k specifikaci jazyka nad Σ.

Operace s jazyky

Uvažujme jazyky L, M. S jazyky lze dělat standardní množinové operace.

Navíc uvažujeme operace

- Zřetězení jazyků. $LM = \{uv \mid u \in L, v \in M\}$.
- Mocnina.

$$L^0 = \{\epsilon\},$$
$$L^n = L^{n-1}L$$

- Uzávěry

$$L^* = \bigcup_{i=0}^{\infty} L^i$$
$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Pro regulární výraz e označíme jemu příslušný jazyk $L(e)$.

- Jeden znak $a \in \Sigma$ je regex. Potom $L(a) = a$.
- Speciální znak . (tečka) je regex, potom $L(.) = \Sigma$.
- pro regexy e, f je ef regex s jazykem $L(ef) = L(e)L(f)$
- pro regex e a speciální znaky $+$, $*$, ? jsou následující regexy a jejich jazyky.
  - $e+$, jazyk $L(e+) = L(e)^+$
  - $e*$, jazyk $L(e*) = L(e)^*$
  - $e?$, jazyk $L(e?) = L(e) \cup \{\epsilon\}$
- pro regexy e, f a speciální znak | je $e|f$ regex s jazykem $L(e|f) = L(e) \cup L(f)$.

Poznámky:

- Pro označování samostatných regexů uvnitř jiného regexu se používají závorky (,).
- V praxi lze speciální znaky do Σ zařadit pomocí zpětného lomítka (např. + je tak \+.)

Příklady (nad ASCII abecedou)

```
regex                   jazyk
-------------------------------------------------------
ahoj                    ahoj
a*                      prazdny retez, a, aa, aaa ...
a|b                     a, b
a|b+                    a, b, bb, bbb ...               vyssi priorita *,+,?
(a|b+)*                 prazdny retez, a, bb, ...       nulta a prvni mocnina
                        aa, abb, abb..., bb....         druha mocnina
.                       a, b, c, d, ...
(.)?                    prazdny retez, a, b, c, d, ..
(.)+                    vsechny retezce mimo prazdneho
(.)*                    vsechny retezce
```

Rozšíření

- spec. znaky pro začátek a konec řádku,

```
^   # zacatek radku
$   # konec radku
```

- třídy znaků

```
[retezec]      # libovolny znak ze slova retezec
[^retezec]     # libovolny znak, ktery není ve slove retezec
[:alnum:]      # libovolny alfanumericky znak
[:digit:]      # libovolna cislice
[:lower:]      # mala pismena
...            # viz napr. manual k utilite grep
```

- specifické počty opakování

```
{n}     # presne n krat
{n,}    # n nebo vicekrat
{,m}    # maximalne m krat
{n,m}   # nejmene n krat, nejvyse m krat
```

- Zpětné reference. Výraz \n, kde n je nenulové celá číslice, představuje slovo, které odpovídá slovu vyhovujícímu n-tému uzávorkovanému podregexu.

```
regex                    jazyk
-----------------------------
(ab)\1                   abab
(.)\1                    aa, bb, cc, dd,
(.)(.)\2\1               aaaa,abba, acca, ...
```

Vyhledávání podle regexu

Je dán text a regex. Vyhledáváme části textu, která jsou slovy v jazyce regexu.

Shoda nemusí být jednoznačná (kvůli opakovačům), nástroje obvykle hledají co nejdelší (vůči opakovačům) shodu. Prázdné řetězce nepovažujeme za nalezenou shodu

```
regex           text               shody
-----------------------------------------
a*b*c*          aaabbbccc          a, b, c, ab, bc,  aab, ...
                                   ..., aaabbbccc
```

GREP

zkratka *global regular expression print*

```
grep [options] regex [files]
```

Prohledá soubory, vybere řádky z těchto souborů, které odpovídají regexu, a vytiskne je na standardní výstup. Pro účely porovnávání se neberou znaky konce řádků. Je-li vynecháno `files`, zpracovává standardní vstup.

- varianta `egrep` má povoluje více metaznaků (pro ty, které jsme si uvedli, už potřebujeme `egrep`), totéž jako `grep -E`. [*doporučuji používat*]
- regex se obvykle píše mezi apostrofy (`'`) nebo uvozovky `"`, aby se zabránilo shellu interpretovat některé metaznaky při expanzi.
- výborně slouží jako filter
- https://www.gnu.org/software/grep/manual/grep.html

Užitečné přepínače

`-s`	nevypisuje chyby o nepřístupnosti, argument je adresář atd.
`-I`	vynechá binární soubory
`-c`	počty výskytů v souborech
`-n`	vypíše čísla řádků
`-i`	nerozlišuje velká a malá písmena
`-v`	vypisuje řádky nevyhovující regexu
`-m k`	umožnuje skončit po nalezení k-té shody
`-R`	rekurzivně se zanořuje do podadresářů
`-h, -H`	nevypisuje/vypisuje jména souborů u shod
`-l, -L`	vypisuje pouze jména souborů, které (ne)obsahují shodu
`-o`	vypisuje na samostané řádky řetězce, které vedli ke shodě

Kódování znaků

- Tradičně (před Unicode): 1 znak = 1 bajt
- ascii tabulka, prvních 128 znaků společných (cca anglická abeceda), zbytek národní abecedy: různorodé. Např. čeština: windows-1250, ISO 8859-2.
- Unicode (32 bitů), UTF-32, UTF-16, UTF-8
- převody pomocí `recode` nebo `iconv`

Konce řádků

- unix: bajt s hodnotou 13 (CR, Carriage Return)
- windows: 2 bajty s hodnotami 10 13 (LF, Line Feed)
- převody: např. pomocí `dos2unix` a `unix2dos`

Základní utility

Bylo v baliku GNU textutils, nyní v GNU coreutils

Zobrazení/spojení/výběr části

```
cat tac, head, tail, cut, paste
```

Filtry

```
rev, sort, uniq, grep
```

Rozdíly v souborech

```
diff, patch
```

```
cat [options] [files]
```

- vytiskne soubory `files` na standardní výstup, po řádcích od začátků souborů
- jsou-li `files` vynechány, použije standardní vstup
- užitečné přepínače
  - `-n` čísluje řádky
  - `-b` čísluje neprázdné řádky
  - `-s` vynechává opakující se prázdné řádky
- varianta `tac` tiskne řádky od posledního

Typická použití

```
cat file.txt file.txt > concatenated.txt
cat -s file.txt > no_blanks.txt
cat file.txt |
```

```
head [options] [files]
tail [options] [files]
```

- tiskne začátky/konce souborů (defaultně prvních/posledních 10 řádků)
- bez parametru bere standardní vstup
- přepínače
  - `-n num` tiskne prvních/posledních `num` řádků
    `head` — `num` záporné, tiskne vše mimo `num` posledních řádků
    `tail` — je-li před `num` znak `+`, vytisne řádky počínající řádkem `num`

Příklad

```
head -n 1 countries.csv
tail -n +2 countries.csv > headless.csv
```

```
cut [options] [files]
```

- Vytiskne vybrané části z každého řádku v souborech `files`, nejsou-li specifikovány žádné soubory, dělá to pro standardní vstup
- řádek lze chápat i jako řádek tabulky, kde jsou sloupce odděleny oddělovačem
- užitečné přepínače
  - `-d delim` určí oddělovač sloupců, defaultně je `TAB`
  - `-f list` vytiskne pouze sloupce z `list`, který je čárkami oddělený seznam čísel sloupců. Pokud řádek neobsahuje oddělovač, je vytištěn celý: toto lze potlačit přepínačem `-s`.
  - `--output-delimiter=string` nastavení oddělovače sloupců ve výstupu (defaultně je vstupní oddělovač)
  - `-b list`, `-c list` tiskne jenom dané bajty či znaky (které jsou opět dány pořadím).

Příklad

```
cut -f 1,2 -d, --output-delimiter=$'\t' countries.csv
```

Pozn: `$'\t'` shell expanduje na znak tabulátoru, cut neumí expandovat `'\t'` sám.

```
paste [options] [files]
```

- spojí jednotlivé řádky (oddělené TAB) z jednotlivých souborů z `files` a vytiskne je na standardní výstup (je to *opačný* program k `cut`)
- přepínač `-d list` specifikuje oddělovače, které se postupně použijí místo TAB.

Příklad

```
cut -f 1 -d, countries.csv > names.txt
cut -f 2 -d, countries.csv > regions.txt
cut -f 3 -d, countries.csv > population.txt
paste names.txt regions.txt population.txt
```

```
rev [options] [files]
```

Pro každý soubor z `files` vytiskne jeho řádky odzadu, bez specifikace `files` zpracuje standardní vstup
Příklad

```
rev names.txt
rev names.txt | rev
```

```
sort [options] [files]
```

- Setřídí řádky ze všech souborů z `files` a v tomto pořadí je vytiskne na standardní výstup. Bez `files` zpracovává standardní vstup. Defaultně lexikální porovnávání.
- přepínače
  - `-m` slití už setřízených souborů
  - `-r` opačné pořadí
  - `-m -H -n -R` typ řazení (měsíce, human-readeable velikosti, numerický, náhodný)
  - `-t znak` oddělovač
  - `-k keydef` definice klíče, např. číslo sloupce (ale lze komplikovaněji, například pozice znaků od do, atd.)

Příklad

```
sort -k 3 -t, -n -r headless.csv
```

```
comm [-123i] file1 file2
```

- Čte soubory (předpokládá lexikální uspořádání) a produkuje tři sloupce jako výstup: řádky pouze ve `file1`, řádky pouze ve `file2`, řádky v obou souborech.
- Přepínače `-1`,`-2` a `-3` potlačují tisk příslušných řádků

Příklad

```
sort countries.csv | head -n -10 > file1.txt
sort countries.csv | tail -n +10 > file2.txt
comm -23 file1.txt file2.txt     # pouze radky unikatni pro file1.txt
comm -13 file1.txt file2.txt     # pouze radky unikatni pro file2.txt
comm -3 file1.txt file2.txt      # ve dvou sloupcich
```

```
uniq [options] [files]
```

- vynechává nebo reportuje opakující se řádky (pouze sousední)
- lze vynechat některé sloupce (chápány jako obsah oddělený bílými znaky)
- není flexibilní, co se týče oddělovačů
- přepínač `-u` vytiskne jenom unikátní řádky, přepínač `-d` vytiskne jenom opakující se řádky (každý jenom jednou)

Příklad

```
tail -n +2 regions.txt | sort | uniq
```

```
diff file1 file2
```

- vyhledá rozdíly mezi dvěma soubory (případně soubory uloženými v adresářových hierarchiích) a vypíše je na standardní výstup (tzv. záplata, v souboru typicky koncovka `.diff`). Záplatu lze potom strojově použít pro přepis `file1` na `file2`.
- záplata v několika formátech (přepínače `-u, -y`)
- tichý chod přepínačem `-q` (návratovou hodnotou oznámí, jestli jsou soubory shodné)
- pro rekurzní použití (= na adresářové struktury) přepínače `-r -N`

```
patch file
```

- aplikuje záplatu na soubor `file`, záplata je očekávána na standardním vstupu, nebo ji lze předat ze souboru pomocí přepínače `-i`

Příklady

```
diff numbers.txt numbers.txt
diff numbers.txt numbers2.txt
diff -y numbers.txt numbers2.txt
diff -u numbers.txt numbers2.txt > z.diff
patch numbers.txt < z.diff
diff numbers.txt numbers2.txt
```

Složitější příklady

Vypište 20tý až 11tý řádek souboru s čísly řádků (v tomto pořadí)

```
cat -n names.txt | tail -n +11 | head | sort -r
```

Vypište první a třetí sloupec pro řádky obsahující země z Evropy, setříděný sestupně podle třetího sloupce, s čísly označujícími pořadí.

```
grep 'EUROPE' headless.csv | cut -f 1,3 -d, --output-delimiter=$'\t' |
sort -n -k 2 -t $'\t' | cat -n
```

(pozn. předchozí je celé jeden příkaz)

Proudový editor sed

- *proudový editor*: řádkový editor, opakovaně
  1. načte řádek do bufferu (pattern space), vynechá přitom znaky konce řádku
  2. test vykonání (typicky regex) a vykonání editovacích příkazů nad vstupním bufferem
  3. vstupní buffer se vypíše (včetně odřádkování), lze potlačit přepínačem `-n`
  4. výmaz vstupního bufferu
- ovlivněné řádky i transformace určujeme při spuštění, potom už editace není interaktivní
- https://www.gnu.org/software/sed/manual/sed.html

```
sed [options] script files
```

- `script` jsou editační příkazy (podrobnosti viz dále)
- `files` jsou editované soubory, jsou-li vynechány, zpracováváme stdin.

Jeden příkaz

```
[addr]command[options]
```

- `addr` určuje, na které řádky se má příkaz aplikovat, jeli vynecháno, bereme automaticky všechny řádky
- `command` jsou akce, které chceme aplikovat

```
sed -n '4p' names.txt
```

- `-n` potlačí tisk řádků
- `4p`: adresa 4 vybíráme 4. řádek, příkaz p řádek pouze vytiskne
- `script` se píše v apostrofech, abychom zabránili shellu v expanzi jmen.

Výběr řádků

- číslem řádku (poslední řádek $)

```
sed -n '4p' names.txt
```

- regexem `/regex/` (pro extended regex přepínač `-E`)

```
sed -n '/Republic/p' names.txt
```

- intervalem řádků: dvě adresy oddělené čárkou, interval je od prvního řádku odpovídajícího první adrese do prvního řádku odpovídajícího druhé adrese. Alternativa druhé adresy: počtem řádků za adresou `+`, do čísla řádku dělitelného číslem `~`.

```
sed -n '4,17 p' names.txt
sed -n '4,/Republic/ p' names.txt
sed -n '4,+10 p' names.txt
sed -n '4,~2 p' names.txt
```

- znak `!` za adresou je negace (adresa tak označuje řádky, na které příkaz neaplikovat)

```
sed -n '4,17! p' names.txt
```

Některé příkazy

Základní jednopísmenné funkce

- q vytiskne pattern space a skončí (celý sed)
- p vytiskne pattern space a pokračuje ve vykonávání (i v aktuálním command)
- d vymaže pattern space a pokračuje další iterací (na dalším řádku)
- n vytiskne pattern space, načte do pattern space nový obsah, pokračuje ve vykonávání (aktuálního příkazu).
- `y/str1/str2/` - v pattern space nahradí znaky z `str1` znaky ze `str2` (na stejných pozicích)
- e interpretuje pattern space jako příkaz shellu a spustí jej, nahradí pattern space jeho výstupem

Substituce

```
s/regex/replacement/flags
```

- nahradí výskyt části odpovídající `regex` pomocí `replacement`, po prvním nahrazení končí (lze změnit pomocí flagu)

```
sed 's/Canada/BearLand/' names.txt
sed -E 's/[A].+/nothing/' names.txt
```

- flag `g`: nekončíme po prvním nahrazení
- flag `number`: nahradíme `number`-tou shodu
- flag `p`: vytiskni pattern space, pokud nastala substituce
- flag `w filename`: po substituci zapiš pattern space do souboru
- flag `e`: po substituci interpretuj obsah pattern space jako příkaz pro shell, ten spusť a jeho výstup použij jako obsah pattern space

```
sed 's/Canada/echo BearLand/e' names.txt
```

Pokročilé substituce

- placeholder `&` pro obsah nalezeného vzoru, lze jej modifikovat pomocí `\U,\u,\L,\l` (změna velikosti prvního/všech písmen)

```
sed -E 's/[A-Z]+/(&)/g' names.txt
sed -E 's/.*/\U &/g' names.txt
```

- zapamatování si částí nalezeného vzoru pomocí `(` a `)`, napíšeme mezi ně části regexu a poté se odkazujeme pomocí `\1` až `\9`.

```
sed -E 's/(^[A-Z][a-z]*) ([A-Z][a-z]*$)/\2 \1/g' names.txt
```

Hold buffer

Místo pro uložení textu (navíc k pattern space)

Funkce

- `h` - nahrazení obsahu hold bufferu obsahem pattern space
- `H` - přidání obsahu pattern space k obsahu hold bufferu jako nového řádku
- `g`, `G` - jako výše, pouze opačným směrem
- `x` - prohodí obsahy hold bufferu a pattern space

Způsoby předání `script`

- jeden přímo z promtu
- více přímo z promptu pomocí přepínače `-e`, pro každý příkaz jeden přepínač
- načtění ze souboru pomocí přepínače `-f`, každý příkaz na novém řádku nebo příkazy odděleny znakem `;`

AWK

Jazyk pro zpracování a analýzu textových dat. Ta lze chápat i jako tabulková data (např. `csv` soubory apod.)

Jméno je zkratkou jmen autorů: A. Aho, P. Weinberger, B. Kernighan.

Jazyk je dán specifikací (POSIX standard), existuje několik implementací, i komerčních. Některé implementace (např. `gawk`) přidaly svoje rozšíření.

O jazyku existuje několik knih, například

- Herold H., awk & sed: Příručka pro dávkové zpracování textu. Computer Press, 2004. ISBN 97880251030 98.
- D. Dougherty, A. Robbins, sed & awk. O'Reilly Media, 1990.

Dále je dokumentace například `man awk`, případně
https://www.gnu.org/software/gawk/manual/gawk.html

Základ fungování

Interpret načítá záznamy (angl. *records*, v základu řádky), ty může rozdělit na políčka (angl. *fields*, v základu oddělena mezerami).

Pro každý řádek aplikuje postupně pravidla. Pravidlo je

- zkontroluje, jestli řádek vyhovoje (např. regex - podobné jako sed)
- pro vyhovující řádky provede akce (= programy)

Akce lze provést i před spuštěním aplikace pravidel na řádky, a po zpracování všech řádků.

Pravidla

```
pattern { action }
```

- `pattern` vynecháno -> bereme vždy shodu
- `action` vynecháno -> tiskneme celý záznam

Spuštění

```
awk [code] [files]
awk -f script_file [files]
```

nebo jako skript s hlavičkou `#! /path/to/awk`

Úvodní příklady (pro představu)

```
awk '{ print }'

awk 'length($0) > 30'

awk '{ if (length($0) > max) max = length($0) }
     END { print max }'

awk 'NF > 0' data/data.txt

awk 'BEGIN { for (i = 1; i <= 7; i++) print int(101 * rand()) }'

ls -l  | awk '{ x += $5 }
              END { print "total bytes: " x }'

awk -F: '{ print $1 }' /etc/passwd | sort

awk 'END { print NR }' data/data.txt
```

Záznamy a políčka

awk nejdříve načte záznam. To dělá tak, že čte znaky ze vstupu, dokud nenarazí na znak konce záznamu. To je defaultně znak nového řádku, lze to ovšem změnit (argumentem utility awk nebo v samotném programu pomocí proměnné RS)

Potom awk rozdělí záznam na políčka, opět pomocí oddělovačů. V základu je oddělovačem libovolná posloupnost bílých znaků. Opět to lze nastavit (i komplikovaně, např. pomocí regexů) z příkazového řádku, nebo s pomocí proměnné FS

Ve skriptu máme k načtenému záznamu a políčkům přístup a to

- pomocí `$1, $2, ...` k jednotlivým políčkům
- pomocí `$0` k celému záznamu
- pomocí proměnné NR k počtu zpracovaných záznamů (včetně aktuálního) (FNR v rámci aktuálního souboru)
- pomocí proměnné NF k počtu políček aktuálního záznamu

Předpokládejme, že máme v souboru `data.txt` údaje o lidech ve formátu

```
Franta Votloukal; Kruta 14; Olomouc; 734092004; Votloukal a syn, a.s.
```

a chceme je převést do hezčí formy. K tomu použijeme skript

```
BEGIN {FS=";"}
{
  print $1
  print $5
  print $2 ", " $3
  print "tel: " $4
}
```

Skript použijeme

```
awk -f adress.awk data/data.txt
```

Shoda se vzorem

Pomocí podmínek (jak jsme viděli v příkladu, více dále) a pomocí regulárních výrazů.

Shoda hledána v celém záznamu

```
/regex/ { akce }
```

Jinak lze použít výraz `expr ~ /regex/`, který hledá shodu `regex` v `expr`, například

```
$3 ~ /regex/ { akce }
```

hledá shodu v třetím políčku

Pokud bychom v předchozím příkladu dali na začátek druhého řádku

```
$1 ~ /^F.*/
```

vypsali bychom pouze lidi, jejichž křestní začínají na F.

Proměnné a aritmetika

Identifikátory: alfanumerické znaky a podtržítko, první znak nesmí být číslo

Hodnota proměnné je číslo a řetězec, používá se podle kontextu (řetězcové hodnoty, které netvoří číslo, mají číselnou hodnotu 0).

Proměnné není nutné deklarovat ani inicializovat, automaticky jsou inicializovány na 0 a prázdný řetězec.

Operátor přiřazení a aritmetické operátory se chovají přirozeně.

`+, -, *, /`	základní aritmetika
`%`	zbytek po celočíselném dělení
`^`	mocnina
`=`	přiřazení
`op=`	zkratka přiřazení a operátoru op

Příklady

1. Počet prázdných řádků v souboru

```
/^$/ {
    x += 1
}
END { print x }
```

2. Pro data typu

```
jarda  10 78 56 34 89
```

vypočtu a vytisknu průměrný zisku bodů

```
{
    total = $2 + $3 + $4 + $5 + $6
    avg   = total / 5
    print $1 " " avg
}
```

Relační výrazy a booleovské operátory

Výraz, který se vyhodnotí na pravdu/nepravdu pomocí vyhodnocení toho, jestli operandy splňují požadovanou relaci.

`<,>,<=,>=`	menší/větší (nebo rovno)
`==`	rovnost
`!=`	nerovnost
`~`	shoda s regexem
`!~`	neshoda s regexem

Kombinace výrazů do složitějších podmínek

`&&`	konjunkce
`||`	disjunkce
`!`	negace

Lze použít místo kontroly shody se vzorem.

Tisk prvního a šestého políčka u záznamů s přesně 6 políčky. Ostatní jsou vynechány.

```
NF == 6 { print $1 ", " $6 }
```

Navíc chceme, aby se třetí políčko shodovalo s regexem

```
NF == 6 && $3 ~ /MA/ { print $1 ", " $6 }
```

U booleovských operátorů je potřeba dávat pozor na prioritu, pro určení priority doporučuji závorky.

Delší příklad

Zpracujeme výpis příkazu

```
    ls -lR
```

Vytiskneme jména všech souborů (1. sloupec) a jejich velikost (2. sloupec), přičemž vynecháme, ve kterém jsou podadresáři. Nakonec vytiskneme statistiku: počet souborů a jejich celkovou velikost.
Na začátku se podíváme na rozdíl výpisů příkazů

```
ls -l
ls -lR
```

Výpis prvního je jednodušší (neobsahuje prázdné řádky a jména poddadresářů při rekurzivním průchodu). Uděláme nejdříve skript pro něj.

Základní skript

```
# tady zpracovavame radky
{
    sum += $5
    count += 1
    print $5 "\t" $9
}

# nakonec vypiseme spoctene statistiky
END {
    print "Total: " sum " bytes (" count " files)"
}
```

Počítáme pouze soubory

```
#pocitame pouze soubory
NF == 9 && /^-.*/ {
    sum += $5
    count += 1
    print $5 "\t" $9
}


END {
    print "Total: " sum " bytes (" count " files)"
}
```

Zpracujeme `ls -lR`

```
NF == 9 && /^-.*/ {
    sum += $5
    count += 1
    print $5 "\t" $9
}

#spocitame pocet podadresaru
NF == 9 && /^d.*/ {
    dirs += 1
}

END {
    print "Total: " sum " bytes (" count " files, " dirs " subdirs)"
}
```

Základy programování v awk

Větvení

```
if (podmínka) {
  #pravda
}
else {
  #nepravda
}
```

`else` část nepovinná, existují varianty formátování.

Příklad: k výpočtu průměrného počtu bodů přidáme sloupec s informací, jestli student prospěl.

```
BEGIN { limit = 50 }
{
    total = $2 + $3 + $4 + $5 + $6
    avg   = total / 5
    if (avg > limit) {
       grade = "pass"
    }
    else {
      grade = "fail"
    }
    print $1 "\t" avg "\t" grade
}
```

Cyklus

```
  while (podminka) {
    # telo cyklu
  }
```

Příklad: výpis čísel od 1 do 10

```
BEGIN {
  i = 1
  while(i < 11) {
    print i
    i += 1
  }
}
```

Úprava výpočtu průměru pro proměnlivý počet písemek

```
BEGIN { limit = 50 }
{
    total = 0
    i = 2
    while(i <= NF) {
       total += $i
       i += 1
    }

    avg   = total / ( NF - 1)
    if (avg > limit) {
       grade = "pass"
    }
    else {
      grade = "fail"
    }
    print $1 "\t" avg "\t" grade
}
```

Asociativní pole

Ukládá vztah klíč a hodnoty (jako `dict` v jazyce Python)

Přistup k poli

```
array[key] = value
array[key]
```

Procházení

```
for (key in array) {
  # tady mame pristup pomoci array[key]
}
```

Test existence klíče

```
key in array
```

Úprava výpočtu průměru

```
BEGIN { limit = 50 }
{
    total = 0
    i = 2
    while(i <= NF) {
      total += $i
      i += 1
    }
    avg   = total / (NF - 1)

    # vlozime prumer do pole s klicem, ktery rovna jmenu
    array[$1] = avg
}
END {
    # projdem vysledky a vypiseme tabulku
    for (name in array) {
      print name "\t" array[name]
    }
}
```

Sportka

```
BEGIN {
    n   = 6
    max = 49
    srand()
    j = 0
    while (j < 6) {
      select = 1 + int(rand() * max)
      while (select in pick) {
        select = 1 + int(rand() * max)
      }
      pick[select] = select
      j += 1
    }

    for (x in pick) {
      print x
    }
}
```

Funkce

Řada zabudovaných funkcí + možnost definovat uživatelské funkce

Některé funkce už jsme viděli např. `srand()`, `rand()`, `length()`, `int()`.

Vybrané zajímavé funkce (na řetězcích)

`gsub(r,s,t)`	v řetězci `t` nahradí každou shodu s regexem `r` řetězcem `s`. Je-li `t` vynecháno, bereme `$0`.
`match(s,r)`	vrátí pozici v `s`, kde nastala shoda s regexem `r` nebo 0, pokud k žádné shodě nedojde. Nastaví globální proměnné `RSTART` `RLENGTH`
`split(s,a,sep)`	rozdělí řetězec `s` na části podle separátoru `sep` a vloží je do pole `a`. Vrací počet vzniklých řetězců. Defaultní hodnota `sep` je `FS`.

Uživatelsky definované funkce

```
function name (parameter-list) {
   body
}
```

Hodnota se vrací pomocí

```
return value
```

Jediné lokální proměnné jsou parametry funkce, proměnné vytvořené v těle funkce jsou globální! Proto se lokální proměnné přidávají do seznamu parametrů, při volání funkce lze některé parametry vynechat.

Příklad: bubble sort

```
function bubble_sort(array, elements,     temp, i, j) {
  i = 0
  while (i < elements) {
    j = 0
    while (j < elements - i - 1) {
      if (array[j] > array[j+1]) {
        tmp        = array[j]
        array[j]   = array[j+1]
        array[j+1] = tmp
      }
      j += 1
    }
    i += 1
  }
}
```

```
function print_array(array, element,  i) {
  i = 0
  while(i < els) {
      printf("%i ", ar[i])
      i += 1
  }
  print ""
}

BEGIN {
  els = 10
  k = 0
  while(k < els) {
    ar[k]  = els - k
    k     += 1
  }

  print_array(ar,els)
  bubble_sort(ar,els)
  print_array(ar,els)
}
```