Unixové systémy a shell

(KMI/XUNIXS)

verze z 13. února 2026

Co je operační systém?

Základní komponenty operačního systému

- *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.

Kdy je OS unixový

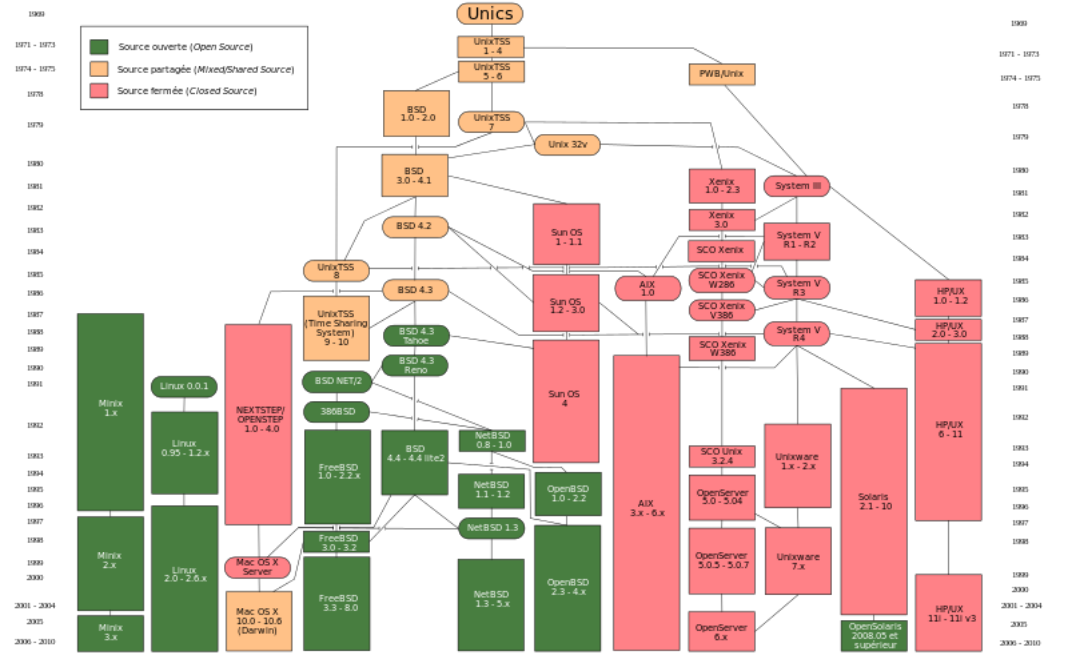
Existuje certifikát, po jehož získání je OS možno obsahovat jako “unix”. Vyžadováno určitý tvar uživatelského a programátorského rozhraní (např. jsou vyžadována existence některých hlavičkových soubourů, shellu a základních utilit atd. (*viz base systém*).

Obecně ale libovolný OS, který koncepčně a technicky vychází z původního unixu a přibližně se drží *unixové filozofie*, např.

- 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

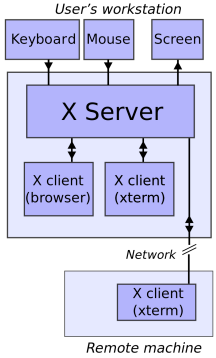
Vznik Unixu a jeho vývoj

- 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


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
BSD 4.3 Reno
System V R4
SCO Xenix W386
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
Solaris 2.1 - 10
HP/UX 6 - 11
Linux 0.95 - 1.2.x
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
AIX 3.x - 6.x
Minix 2.x
FreeBSD 3.0 - 3.2
NetBSD 1.3
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
NetBSD 1.3 - 5.x
HP/UX 11i - 11i v3
Minix 3.x
Mac OS X 10.0 - 10.6 (Darwin)
FreeBSD 3.3 - 8.0
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)



Linuxové distribuce

- base system (linuxové jádro a utility, často GNU)
- 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, OpenSuse, Ubuntu, Debian, Arch, Gentoo, , Mint,

Základy shellu

příkazová řádka, interpretuje a provádí příkazy, tiskne výsledek

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

Základy ovládání	
zápis příkazu	zápis z klávesnice
spuštění příkazu	↵
doplňování	⇤⇥
skok na začátek řádku	Ctrl + a
skok na konec řádku	Ctrl + e
vyjmutí (od kurzoru do konce řádku)	Ctrl + k
vyjmutí (celého řádku)	Ctrl + u
vložení	Ctrl + y

Příklady příkazů

```
pwd         # vytiskne (abs. cestu) aktuálního adresáře
ls          # vylistuje obsah aktuálního adresáře
echo        # tiskne argumenty na standardní vstup
exit        # ukončí aktuální shell
whoami      # vytiskne aktuálního uživatele
hostname    # vytiskne jméno počítače
uname       # tiskne systémové informace
who         # tiskne seznam přihlášených uživatelů
```

Obecný tvar příkazu

```
cmd [přepínače] [arguments]
```

(*Hranaté závorky v dokumentaci příkazů značí, že danou věc lze vynechat*)

Syntax přepínačů (obvykle)

```
-l                # krátký
--slovo           # dlouhý
```

Například

```
ls -l --color
```

Typ příkazu

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

Zjišťování toho, kde hledat informace

```
type          # zjistí typ příkazu
apropos       # prohledá jména a popisy manuálových stránek
whatis        # najde manuálové stránky
which         # najde umístění binárního souboru
whereis       # najde umístění binárního souboru a dokumentace
```

Pager

Program `man` je přepne terminál do celoobrazovkového režimu. Podobným programům, které tak činí za účelem zobrazení nějakého obsahu, říkáme pagery (z angličtiny, zobrazují po stránkách). Má specifické ovládání (část je společná všem pagerum, často nápověda `h`, ukončení `q`, vyhledávání `/`)

Příkladem dalšího pageru je program `less`, který zobrazuje obsah souboru nebo svůj standardní vstup.

Uživatelské nastavení bash

Pro nás postačuje soubor `.bashrc` v domovském adresáři.

Příklad: konfigurace promptu

- proměnná PS1
-

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

- `PS1='\n[ \u@\h: \w ]\n$ '`

Proměnné prostředí

bez typu, bere se jako posloupnost znaků

Tisk proměnné

```
echo $var
```

(*Toto je první příklad expanze: procesu, kterým shell mění zadaný příkaz. Nejdříve* `$var` *nahradí hodnotou proměnné* `var` *a až poté spustí* `echo`.)

Zajímavé (systémové) proměnné

```
PS1           # prompt první úrovně
HOME          # domovský adresář aktuálního uživatele
PWD           # aktuální adresář
OLDPWD        # předchozí adresář
SHELL         # defaultni shell
PATH          # adresáře pro vyhledávání spustitelných souborů
```

Alias

```
alias name=command
```

- 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
```

Historie příkazů

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

:parametr
n           zobrazi poslednich n polozek v historii
```

V shellu

```
!n                  n-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), potom ↵ je provedení nalezeného příkazu a Ctrl + g ukončení prohledávání.

Přesměrování vstupu a výstupu

Programy mají následující vstup a výstup (mimo soubory aj., které si otevřou sami)

- standardní vstup (`stdin`)
- standardní výstup (`stdout`)
- standardní chybový výstup (`stderr`)

Tyto jsou typicky klávesnice a terminál. Lze je přesměrovat do/ze souboru, případně jinému programu.

Přesměrování stdout	
`> file`	do souboru, existující soubor je přepsán
`>> file`	do souboru, obsah přidán na konec souboru
`program-A | program-B`	stdout programu `A` je přesměrován na stdin programu `B`.

Příklady

```
# vytvoří (nebo přepíše, pokud existuje) soubor foo.txt,
# jeho obsahem je hodnota proměnné PATH
echo $PATH > foo.txt

# připojí k souboru další řádek obsahující xxxxx
echo xxxxx >> foo.txt

# co je po provedení následujícího řádků obsahem souborů foo.txt a files.txt?
echo foo.txt > files.txt

# program cat (od concatenate) tiskne obsah souborů za sebe na stdout
cat files.txt foo.txt
```

Filtry

Programy často umí zpracovat obsah souboru nebo standardní vstup, pokud není soubor zadán. Takové program svůj vstup nějak transformuje, výsledek vypisuje na stdout. Dá se na to dívat tak, že svůj vstup filtruje.

Příklad: program wc

```
# vytiskne počet řádků v souboru foo.txt
wc -l foo.txt

# také vytiskne počet řádků v souboru foo.txt s použitím přesměrování
cat foo.txt | wc -l

# vytiskne počet položek v aktuálním adresáři
ls -1 | wc -l
```

Přesměrování stdin	
`< file`	obsah souboru je dán na stdin programu
`<< END`	načítám více řádků na stdin, končím řádkem s obsahem END

```
# spočítá počet řádků v souboru file.txt
wc -l < file.txt

# vytiskne dva řádky
cat << END     # enter, na každém novém řádku se zobrazí druhotný prompt $$
$$ ahoj         # enter
$$ svete        # enter
$$ END          # enter

# s přesměrováním do souboru
cat << DONE > fruit.txt
$$ apple
$$ orange
$$ DONE
```

- pro přesměrování `stderr` použijeme 2>, např. `cmd 2> /dev/null` přesměruje stderr `cmd` do soubouru `/dev/null`.
- přesměrování `stderr` na `stdout`: `2>&1`
- přesměrování `stderr` a `stdout` do stejného souboru: `&> file`
- přesměrování `stdout` a `stderr` do roury: `|&`

Úkoly z první přednášky

1. Získejte přístup k fungující instalaci některé Linuxové distribuce. (Třeba nainstalovat do *VirtualBoxu*; odkaz ke stažení a návod na instalaci jsou na webu předmětu.) Distribuce může být libovolná, jako distribuce pro začátečníky bývají označovány: *Linux Mint, Ubuntu, Pop!_OS, Zorin OS, Fedora XFCE spin*.
2. Spusťte emulátor terminálu. Projděte si slajdy a vyzkoušejte uvedené příkazy a příklady. Experimentujte s vlastními příkazy (nezapomeňte, že máte k dispozici manuálnové stránky).
3. Volitelně se v systému zorientujte, vyzkoušejte si nainstalovat nějaký program, nastavit barvy a font v emulátoru terminálu.

Souborový systém

Co je souborový systém

Logický způsob uspořádání uložení dat na (částech) ukládacích zařízení do jedné stromové struktury,

Hlavní objekty

- soubor: atomický kus uložených dat (z logického pohledu)
- adresář: kolekce souborů a jiných adresářů (lze chápat jako speciální soubor obsahující tuto kolekci).
- vztah rodič a potomek (je-li A v adresáři B, pak je jeho potomkem)

Vše je v jednom adresáři, je to kořenový adresář, jeho jméno je /.

Cesta

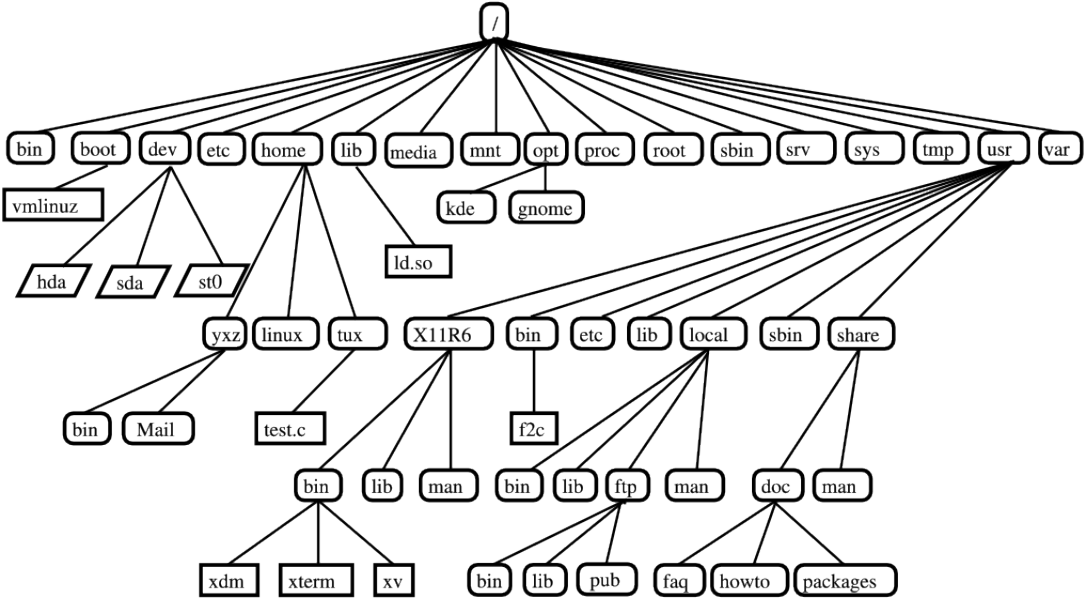
- relativní (cesta stromem mezi dvěma obecnými uzly)
- absolutní (cesta stromem od kořene do nějakého uzlu),
- jednotlivé uzly odděleny lomítkem (znakem /),
- aktuální adresář `.`, nadřazený adresář `..`

Příklady

```
# viz obrázek na dalším slajdu

# absolutní cesta
/usr/local/bin

# relativní cesta z adresáře /home/tux do adresáře /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

Další význam pojmu souborový systém: implementační pohled

- konkrétní způsob správy uložení dat na médiu
- EXT[2,3,4], (V)FAT, NTFS, ZFS, Btrfs atd.

Inode, linky

- *inode*
  Struktura reprezentující soubor, obsahuje metainformace (typ souboru, práva, data změny atd.) a ukazatel na data. Každý inode má číslo.
- *hardlink*
  Jméno a inode číslo
- *symlink, symbolický link*
  Soubor obsahující absolutní cestu (i k adresáři)

Ve stromové hierarchii jsou vlastně hardlinky a symlinky. Inode pro soubor může mít více hardlinků, má-li 0 hardlinků, je odstraněn. Inode pro adresář má pouze jeden hardlink (kvůli acykličnosti grafu souborů). Pokud neexistuje objekt, na který odkazuje cesta v symlinku, není symlink smazán. Adresář je soubor obsahující jména a inode čísla svých položek.

```
# zobrazení inode čísel jednotlivých položek
ls -i

# vytváření linků: ln [option] TARGET LINK_NAME
# vytvoření hardlinku
ln /home/osicka/foo.txt blah.txt

# vytvoření symlinku
ln -s /home/osicka/foo.txt sym-blah.txt
```

Globbing

Shell provádí expanze výrazů. Pro teď si expanzi představme jako náhradu části textu v příkazové řádce nečím jiným. Expanze tak transformuje obsah příkazové řádky. Samotný příkaz je spustěn až po této transformaci.

Nahrazování se někdy provádí tak, abychom po nahrazení dostali platné cesty (typicky v případech, kdy shellu necháváme v nahrazování větší volnost).

Nahrazovaný výraz	Význam
`*`	0 nebo více libovolných znaků
`?`	jeden libovolný znak
`[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, plná expanze, nemusí být cesta!
`[^chars]`	jakýkoliv znak, který není v chars
`~`	zkratka domovského adresáře

Když není expanze nalezena, je znak chápán doslovně.

Expanze se neprovádí v části mezi apostrofy. V části mezi uvozovkami se expandují pouze jména proměnných.

Lze hezky testovat pomocí `echo`, které tiskne na stdout přesně takový řetězec, který dostane na argument.

```
# příkaz                        # co vytiskne (v aktuálním adresáři)
# ------------------------------------------------------------------
echo *                          # obsah adresáře
echo *.txt                      # soubory (a adresáře) s koncovkou .txt
echo [abc]*                     # soubory a adresáře začínající jedním z a,b,c
echo /home/osicka/{doc,bin}     # /home/osicka/doc /home/osicka/bin
```

Základní příkazy

`cd`	změna aktuálního adresáře
`mv`	přejmenování, přesun
`cp`	kopie
`rm`	smazání
`mkdir, rmdir`	vytvoření, smazání prázdného adresáře
`ls`	výpis obsahu adresáře
`ln`	vytváření linků (hardlinků i symlinků)
`du`	velikosti adresářů a souborů
`file`	zjištění typu souboru
`touch`	vytvoření/změna času modifikace souboru
`df`	informace o volném/obsazeném místě

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`). Existuje 8 typů přístupu odpovídající podmnožinám `{r,w,x}`.
- Význam práv

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

Zobrazení práv

```
ls -l filename # níže je vysvětlen výstup

-rw-r--r-- 1 osicka osicka 74815 Sep  9 13:40 slides.tex
|[-][-][-]   [----] [----]
| |  |  |       |      |
| |  |  |       |      +------------> skupina
| |  |  |       +-------------------> vlastník
| |  |  |
| |  |  +-----------------------------> práva ostatních
| |  +--------------------------------> práva skupiny
| +-----------------------------------> práva vlastníka
+-------------------------------------> filetype: - file
                                                  d dir
                                                  l link

# alternativně pomocí utility stat
stat filename            # kompletní info o souboru
stat -c "%A" filename    # práva jako u ls -l
stat -c "%a" filename    # práva jako číslo
```

Změna práv

```
chmod permissions files
```

Zadání `permissions`, první verze

```
xyz
```

- x řetězec ze znaků `u`, `g`, `o`, určuje, komu budou práva změněna. (Lze použít znak `a` s významem všichni.)
- y je změna práv: `+` je přidání, `-` je odebrání a `=` je nastavení práva,
- z je řetězec ze znaků `r`, `w`, `x`, označuje, o která práva se jedná.

```
chmod g-w,o-r file.txt            # skupine odebere zápis, ostatním čteni
chmod u=rwx,g=r,o=  file.txt      # vlastniku nastavíme vše, skupině čtení,
                                  # ostatním nic
```

Zadání `permissions` pomocí čísla

- Typům práv přiřadíme trojbitová čísla `r -> 4 (100), w -> 2 (010), x -> 1 (001)`.
- Práva vyjadříme číslicí, která vznikne součtem `{r,x}` je `5 (101)`. Součet lze dělat i po bitech. Argument parameters jsou tři číslice, v pořadí `ugr`.

```
chmod 750 file.txt     # vlastnik vše, skupina čtení a spuštění, ostatní 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
```

Defaultní práva

Pro nově vytvořené soubory 666, pro nově vytvořené adresáře 777. Lze změnit pomocí masky, kterou lze zjistit a nastavit pomocí utility `umask`.

Maska je číslo, které vyjadřuje *neudělená práva* (nelze jí tedy práva přidávat). Když odečteme masku od 666 dostaneme defaultní práva pro soubory, když ji odečteme od 777, dostaneme defaultní práva pro adresáře.

```
je-li maska 022, potom jsou defaultní práva

soubor   666-022 = 644   tj. rw-r--r--
adresář  777-022 = 755   tj. rwxr-xr-x

potřebnou masku pro soubory tak spočítáme pomocí 666 - cílená práva
v našem příkladu tedy

maska = 666 - 644 = 022
```

Hledání souborů - program find

```
find [options] [where] [pattern]

# options - různá nastavení, např. jak se chovat k linkům
# where   - v jakém adresáři hledat, hledá se v celém podstromu
# pattern - výraz, kterým určíme, jaké soubory hledat a co s nimi dělat
```

Je-li prázdné `where`, bereme aktuální adresář. Pokud `pattern` neobsahuje podmínky pro hledání, vyhovuje podmínkám celý obsah `where` (celý podstrom). Není-li specikováno, co s výsledky dělat, jsou vytištěny. Přitom výsledek je cesta k souboru nebo adresáři vyhovujícímu podmínkám.

```
# vytiskne obsah podstromu, jehož kořenem je aktuální adresář;
# pro každý soubor a adresář jeden řádek obsahující relativní cestu k němu
find
```

```
# příklady vyhledávacích podmínek

# soubory s koncovkou .conf
find /etc/ -name "*.conf"

# soubory větší nez 400 MB; + znamená větší, - menší, nic přesně.
find ~ -size +400M

# typ souboru: soubor f, adresář d, další v manuálu
find ~ -type f

# soubory změněné v posledních pěti dnech
find ~ -type f -mtime 5

# soubory, jejichž vlastníkem je uživatel osicka
find ~ -user osicka

# soubory, které nekončí koncovkou .conf
find ~ -type f -not -name "*.conf"
```

Co dělat s vysledky

```
-delete
-exec command             # pro každou shodu spustí command
-ok   command             # jako exec, ptá se na potvrzení
-print format             # formátovaný výstup
```

```
# v command je nalezena cesta značena pomocí {}, navíc musí končit středníkem,
# který nezpracuje shell (napíšeme \; nebo ';')

# přesune všechny .jpg soubory z posledních pěti dnů do adresáře ~/pictures
find -name "*.jpg" -mtime 5 -exec mv {} ~/pictures/ \;

# vytiskne k nalezeným souborům i velikosti v kilobajtech
find . -type f -printf '%p\t%k KB\n'
```

Další užitečné options a podmínky

```
# options
-L            # při vyhledáván se vnořujeme symbolických linků
-D            # zobrazuje pomocné informace

# podmínky
-maxdepth n   # strom je prohledáván pouze do hloubky n
-mindepth n   # strom je prohledáván od hloubky n
```

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`

Úkoly z druhé přednášky

Obecný úkol: Vyzkoušejte si základní práci s příkazy z přednášky. Konzultujte manuálové stránky (nebo jinou dokumentaci, např. online).

Pomocné úkoly (některé převzaty z prezenčního studia od doc. Outraty).

- Zobrazte cestu k domovskému adresáři (už to umíte 2 způsoby).
- Zobrazte velikost aktuálního adresáře.
- Vytvořte adresář `test` a do něj textový soubor `test.txt` obsahující nějaký text (např. pomocí textového editoru, případně pomocí přesměrování). Obsah souboru zobrazte v konzoli.
- Vytvořte adresář `test2` a soubor `test/test.txt` do něj přesuňte.
- Do adresáře `test` vytvořte symbolický link na soubor `test2/test.txt`. Pomocí tohoto linku zobrazte jeho obsah.
- Soubor `test2/test.txt` smažte a pomocí linku v adresáři `test` zobrazte jeho obsah. Co se stalo?

- Vyhledejte soubory v adresáři `/etc/` větší než 1 kB a a současně nekončící koncovkou `.conf`.
- Napište příkaz, který dá na stdout cesty ke všem souborům (ne adresářům) se jménem začínajícím `IMG` a koncovkou `jpg`
- Napište příkaz, který dá na standardní výstup cesty ke všem souborům (nikoliv adresářům), rekurzivně v domovském adresáři, modifikovaným ne dříve než před 10 dny.
- Napište příkaz, který všechny soubory z adresáře `foo`, ale nikoliv z jeho podadresářů, obsahující ve jméně `xx`, přesune do adresáře `bar`.
- Vytvořte soubor `test.txt`. Odeberte vlastníkovi právo zápisu a poté do něj zkuste zapsat. Práva souboru vraťte.
- Vytvořte adresář `foo`. Nastavte všechna práva vlastníkovi a pouze právo vstupu skupině a ostatním uživatelům.

Procesy

Proces je aktivní entita zhruba odpovídající běžícímu programu

- má přiděleny zdroje: procesorový čas, pamět, otevřené soubory, síťová spojení.
- OS udržuje jeho stav, rozhoduje a rozhoduje přidělování jiných zdrojů.
- Existují ještě vlákna.

Strom procesů, pid

- `pid` - jednoznačný identifikátor procesu (zkratka *process id*)
- proces spouští uživatel (technicky ale jiný proces)
  - kopie spouštěcího procesu (`fork`)
  - 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: klasicky `init`, má `pid` rovno 1.

Přistupová práva procesů

- Proces má vlastníka a skupinu, od kterého dědí přístupová práva. Obvykle je to 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`.

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.

Monitorování procesů

```
ps
# nejchaotictější systém přepínačů všech dob (z historických důvodů)
# -o pid,ppid,cmd ...
# 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      # pager, vlákna: n*[{command}]
htop     # pager, novější (proces ~ task)
pstree   # strom procesů na stdout
pgrep    # vyhledání podle vzoru (např. jména, více později).
```

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, (SIGTSTP)
bg job-number   # rozbehnuti na pozadi
fg job-number   # presun do popredi
```

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

Ukončení procesu

```
kill -NAME pids   # NAME je jméno signálu, jde bez SIG
```

- obecný mechanismus posílání signalů procesům, některé proces může zpracovat podle svého
- signál `SIGKILL`: okamžité ukončení procesu, proces nemůže reagovat (až na `init` proces), signál `SIGTERM` žádost o terminaci, proces může reagovat, signál `SIGINT` přerušení procesu (velmi podobné `SIGTERM`), často navázáno na ctrl + c (v shellu).

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)
- `nohup`: proces běží i po odhlášení uživatele, `disown`. Za obojím je změna rodiče.
- *zombie proces* - potomek, který skončil, ale OS jej udržuje v záznamech, aby si rodič mohl vyzvednou návratovou hodnotu.

Úkoly ze třetí přednášky

- Vyzkoušejte si monitoring procesů.
- Zjistěte všechny procesy aktuálního uživatele.
- Spusťte proces, např. `sleep`, zastavte jej, převeďte jej na pozadí, a poté opět na popředí.
- Spustťe proces, a ukončete jej pomocí příkazu `kill`, spusťte jej znovu a ukončete jej z programu `htop`.

```c
/*
 * compile, run, and then search for the undead with
 * > ps axo stat,ppid,pid,comm | grep '^Z'
 */
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main()
{
        pid_t child_pid = fork();
        if (child_pid > 0)
        {
                sleep(60);
        }
        else
        {
                exit(0);
        }
        return 0;
}
```

Práce s textem

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`

Textové editory - vi

Obvykle je všude nainstalován.

Pracuje v režimech (tj. je modální). Dva základní režimy

1. *Command mode* - příkazový řežim
   - `vi` se v něm spouští
   - pro pohyb v textu, mazání, kopírování, ukládání a ukončení.
   - návrat z jiného módu pomocí Esc.
2. *Insert mode* - vkládací režim
   - pro psaní.
   - Aktivuje se stiskem `i`, a nebo `o`.

Základní příkazy

Pohyb

- `h, j, k, l` – vlevo, dolů, nahoru, vpravo
- `0` – začátek řádku
- `$` – konec řádku
- `gg`/`G` – začátek/konec souboru

Úpravy

- `x` – smazat znak
- `dd` – smazat řádek
- `yy` – kopírovat řádek
- `p` – vložit

Ukládání a hledání

- Uložit změny: `:w`
- Ukončit editor: `:q`
- Uložit a ukončit: `:wq`
- Ukončit bez uložení: `:q!`
- Hledání slova: `/slovo`

Textové editory - Emacs

Spuštění v terminálu: `emacs -nw file`

Otevření a uložení

- ctrl + x ctrl + f – otevřít soubor (find file)
- ctrl + x ctrl + s – uložit soubor (save)
- ctrl + x ctrl + c – ukončit Emacs

Pohyb v textu

- ctrl + f – vpřed (forward)
- ctrl + b – vzad (back)
- ctrl + n – dolů (next)
- ctrl + p – nahoru (previous)

Úpravy textu

- ctrl + d – smazat znak pod kurzorem
- meta + d – smazat slovo (na současné klávesnici je Meta = Alt)
- ctrl + k – smazat zbytek řádku
- ctrl + y – vložit poslední smazaný text (yank)
- ctrl + / nebo ctrl + x u – zpět (undo)

Hledání a nahrazování

- ctrl + s – hledat dopředu (incremental search)
- ctrl + r – hledat dozadu
- meta-% – nahradit text (query replace)
- ! v režimu nahrazování – potvrdit všechny výskyty

Textové editory - další

- nano
- jed
- joe
- pico

Regulární výrazy

Uvažujeme abecedu znaků Σ neobsahující rezervované znaky.

Slovo 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 (posloupnost 0 znaků) označujeme ϵ.

Příklad: Abeceda $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}\}$, `aaabc` je slovo nad Σ. Množina {abc, aa, aaab, abc} nebo množina všech slov začínajících znakem a jsou jazyky.

Regulární výraz je slovo nad abecedou obsahující Σ obohacenou o rezervované znaky, vytvořené podle následujících pravidel.

- Jeden znak $a \in \Sigma$ je regex.
- Speciální znak . (tečka) je regex. (. je speciální znak)
- Jsou-li e, f regulární výrazy, je jimi i ef a $e|f$. (| je speciální znak.)
- Je-li e regulární výraz, jsou jimi i $e+$, $e*$, $e?$. ($+, *, ?$ jsou speciální znaky.)

Dále za speciální znaky považujeme závorky, do nichž lze regulární výraz vložit.

Operace s jazyky

Uvažujme jazyky L, M. S jazyky lze dělat standardní množinové operace (Jazyk je množina slov.)

Další operace

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

  Příklad: $L = \{aa, bb, cc\}$, $M = \{ab, c\}$, $LM = \{aaab, aac, bbab, bbc, ccab, ccc\}$.

- *Mocnina*. $L^0 = \{\epsilon\}, \quad L^n = L^{n-1}L$

  Příklad: $L = \{a, bc\}$, $L^2 = \{aa, abc, bca, bcbc\}$
  $L^3 = \{aaa, aabc, abca, abcbc, bcaa, bcbc, bcbca, bcbcbc\}$

- *Uzávěry*. $L^* = \bigcup_{i=0}^{\infty} L^i, \quad L^+ = \bigcup_{i=1}^{\infty} L^i$.

  Příklad: $L = \{aa, b\}$,
  $L^* = \{\epsilon, aa, b, aaaa, aab, baa, bb, aaaaaa, \dots\}$
  $L^+ = L^* - \{\epsilon\}$

Sémantika regulárních výrazů

Regulární výraz indukuje jazyk. Jazyk indukovaný regulárním výrazem r značíme $L(r)$. Je definován následovně.

- Pro $a \in \Sigma$ máme $L(a) = a$.
- $L(.) = \Sigma$.
- Pro regulární výrazy e, f máme

  $L(ef) = L(e)L(f)$ a
  $L(e|f) = L(e) \cup L(f)$.

- Pro regulární výraz e máme

  $L(e+) = L(e)^+$,
  $L(e*) = L(e)^*$,
  $L(e?) = L(e) \cup \{\epsilon\}$

Příklady

`*`, `+`, `?` mají vyšší priority než `|` a zřetězení.

```
reg. výraz                jazyk
----------------------------------------------------------
a                         a
ab                        ab
a|b                       a, b
a|b+                      a, b, bb, bbb ...
(a|b+)*                   prazdny retez, a, b, bb, ...
.                         a, b, c, d, ...
(.)?                      prazdny retez, a, b, c, d, ..
(.)+                      vsechny retezce mimo prazdneho
(.)*                      vsechny retezce
```

Rozšíření

V unixových systémech (i jinde) existují rozšíření regulárních výrazů (detaily často závisí na konkrétním programu, programovacím jazyku). Níže jsou některé příklady, detaily lze nalézt v manuálu ke konkrétnímu programu (např. `grep`).

- spec. znaky pro začátek a konec řádku,

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

- třídy znaků (příklady)

```
[retezec]      # libovolny znak ze slova retezec
[^retezec]     # libovolny znak, ktery není ve slove retezec
[[:alnum:]]      # libovolny alfanumericky znak
[[:digit:]]      # libovolna cislice
[[:lower:]]      # mala pismena
```

- 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 regulárního výrazu

Vyhledáváme části textu, která jsou slovy v jazyce indukovaném regulárním výrazem.

Shoda nemusí být jednoznačná (kvůli opakovačům), programy pro vyhledávání obvykle hledají co nejdelší (vůči opakovačům) shodu. Často také vyhledávají shody, které se nepřekrývají.

Program grep

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

Prohledá soubory, vybere řádky z těchto souborů, které odpovídají regexu, a vytiskne je na standardní výstup.

- 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.

Procvičit a testovat regulární výrazy lze pomocí

```
grep -E --color=auto 'regex'
```

Poté lze zadávat řetězec. Po ↵ `grep` vypíše a barevně zvýrazní shody, které v řetězci našel a očekává další vstup. (Konec pomocí ctrl + d .)

Pokud chceme použít metaznak, jako obyčejný znak, musíme jej *backquotovat* (tj. napsat před něj zpětné lomítko). Například `\.` je obyčejný znak tečka.

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

```
# Další příklady

# čísla (s volitelnou desetinnou částí s tečkou)
[0-9]*\.?[0-9]+

# povolíme i desetinnou čárku
[0-9]*(\.|,)?[0-9]+

#formát telefonního čísla v ČR
(\+?420)? ?[0-9]{3} ?[0-9]{3} ?[0-9]{3}

#identifikator v C
[_[:alpha:]][_[:alnum:]]*
```

Úkoly ze čtvrté přednášky

1. Napište příkaz, který vypíše čísla prázdných řádků v souboru.
2. Napište příkaz, který na standardní vstup vypíše počet řádků v souboru, které neobsahují číslici.
3. Napište příkaz, který vytiskne řádky textového souboru, které obsahují dvě stejná přirozená čísla oddělená mezerou (a už nic jiného).

Základní programy pro zpracování textu

Zobrazení/spojení/výběr části

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

Filtry

```
rev, sort, uniq, tr
```

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 opakujícící 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 |
```

Program `split` je v jistém smyslu inverzní, dělí jeden soubor na více souborů.

```
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
```

```
tr string1 [string2]
```

Čte stdio, zapisuje stdout. Při nejjednodušším použití zamění symboly ze `string1` za symboly na stejných pozicích ve `string2`. (Nebo je smaže). Umí také smrsknout posloupnosti tvořené jedním symbolem na jeho jedno opakování a komplikovanější zadání transformace.

```
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. sloupce od,do (ale lze komplikovaněji, například pozice znaků od do, atd.)
  - `-u` vynechává řádky obsahující klíče, které už v souboru byly.

Příklad

```
sort -k 3,3 -t, -n -r headless.csv   # setřízeno podle třetího sloupce
sort -k 2,2 -t, -u headles.csv       # unikátní hodnoty v druhém sloupci
```

```
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)

Úkoly z páté přednášky

1. Vyextrahujte druhý sloupec ze souboru `data.csv`, kde jsou sloupce odděleny čárkou.
2. Zobrazte řádky, které jsou společné pro dva soubory.
3. Spočítejte výskyt jednotlivých slov v souboru (ignoruj velikost písmen). *Nápověda: nahrazení nealfabetických znaků znakem nového řádku.*
4. Získejte všechny IP adresy ze souboru access.log (předpoklad: IP je na začátku řádku, oddělené od zbytku řádku mezerou).
5. Vypište 5 nejčastěji se vyskytujících e-mailových domén v souboru, kde je na každém řádku jedna mailová adresa (a už nic jiného).

sed & awk

Proudový editor sed

- *O*pakovaně provádí
  - načte řádek do bufferu, tzv. *pattern space*, vynechá přitom znaky konce řádku
  - test jestli vykonat a vykonání editovacích příkazů nad bufferem
  - buffer se vypíše (včetně odřádkování), lze potlačit přepínačem `-n`
  - výmaz bufferu
- ovlivněné řádky i transformace určujeme při spuštění, potom už editace není interaktivní
- https://www.gnu.org/software/sed/manual/sed.html

Ukážeme si princip a nejdůležitější editační příkaz.

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

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

Dále lze `script` předat

- 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 `;` (to může záležet na konkrétním příkazu).

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
```

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 9788025103098.
- 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. *record*s, v základu řádky), ty může rozdělit na políčka (angl. *field*s).

Pro každý záznam aplikuje postupně pravidla. Pravidlo se sestává z

- kontroly, jestli záznam vyhovuje,
- programu, který se provede, pokud záznam při kontrole vyhověl.

Akce lze provést i před spuštěním aplikace pravidel na záznamy, a po zpracování všech záznamů.

Pravidla

```
pattern { action }
```

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

Spuštění

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

Úvodní příklady

```
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()) }'

awk '{ x += $2 }
     END { print "total " 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 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 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.

Větvení

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

`else` část nepovinná.

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)
}
```

Úkoly z šesté přednášky

1. Změňte všechna slova ”foo”na ”bar”.
2. Nahraďte druhý výskyt slova ”error”v každém řádku za ”warning”.
3. Odstraňte znak ; na konci řádku (pokud tam je).
4. Uvažte soubor s následujícím obsahem

```
Alice     95     1200
Bob       78     800
Charlie   105    1500
David     67     950
Eve       88     1100
ERROR     0      0
Frank     101    2000
Grace     75     600
```

1. Vypište průměr hodnot třetího sloupce, u řádků kde je hodnota v druhém sloupci větší než 80.
2. Odstraňte řádky, kde je první sloupec `ERROR`.
3. Přidejte sloupec s rozdílem mezi třetím a druhým řádkem.