### Vývoj verzovacích systémov

Potrebujem sledovať zmeny súborov v čase – ideálny nástroj je verzovací systém.

Pomocou verzovacieho systému mám podchytené zmeny a zároveň sa môžem kedykoľvek vrátiť k ľubovoľnej predošlej verzii. Systém správy verzií je označovaný jako VCS z anglického Version Control Systém.

Na zaciatku vývoja vznikli lokálne VCS. Nevyhodou bolo, že keď odišiel počítač, kde bola hlavne repository, bola celá história verzovania v prdeli… Najznámejsí VCS bol rcs



Ďalším evolučným krokom boli centralizované systémy správy verzií. Repozitory obvykle niekde na serveri, ostatné PC jako klienti. CVCS z angl. Centralized Version Control Systém. Známe systémy: CVS, Subversion, Perforce alebo StarTeam. Problém je podobný jako s VCS. Pokial skolabuje server, stratím všetky verzie projektu.



Aktuálnym stupienkom vývoja sú distribuované verzovacie systémy, kde každý účastník projektu ma kompletnú historiu verzií. DVCS z angl. Distributed Version Control Systém – príklady: Git, Mercurial, Bazaar nebo Darcs

To je potřebné mať stále na pamäti, že každý má úplne všetko – nie je to len snapshot – aktuálna verzia scriptov - mám tam naozaj všetko.



### História systému Git

Systém Git vznikol s dôvodu verzovania zdrojových kódov jadra sytému linux. V rokoch 1991-2001 bolo jadro Linuxu spravované formou záplat a archívnych súborov. V roku 2002 sa začal využívať komerčný systém BitKeeper. V roku 2005 sa zhoršili vztahy medzi vývojarmi a spoločnosťou vyvýjajúcou BitKeeper – odmietli ho ďalej poskytovať zdarma. To podnietilo vývojárov a obzvlášť Linusa Torvaldsa, tvorcu Linuxu, aby vyvinuli vlastný nástroj, založený na poznatkoch, které nazbierali pri používaní systému BitKeeper. Rok 2005 je rokom vzniku Git-u.

### Základné principy fungovania

Jednotlive systémy sa môžu odlišovať tým jako historizujú. Napríklad Mercurial ukladá prírastky, čo sa zmenilo – změna – často označovaná gréckym písmenkom delta: **∆**



GIT ukladá aktuálnu podobu zmenenných súborov. V záujme úspory miesta v prípade, že sa na určitom súbore nič nezmenilo, uloží v novej verzii len referenciu na predošlú verziu nezmeneného súboru.



Podobne jako aj Mercurial pracuej GIT lokálne. Takže môžem plnohodnotne pracovať aj bez pripojenia k iným počítačom, alebo k internetu.

Git si pre každý súbor vytvára kontrólny súčet, ktorý sa následne kontroluje pri každej akcii. Konkrétne ide o SHA-1 hash. Je to 40 hexadecimalnych znakov. Git neukladá pod názvom súboru, vždy pouzije SHA hash.

DVCS si ukladá jednotlivé zmeny a skôr je problém niečo odmazať, než problém že by sa malo niečo stratiť.

### 3 základné stavy spravovaných súborov

Projekt je v systéme Git rozdelený do troch hlavných častí:

1. adresár systému Git (Git directory) – miesto kde Git umiestňuje metadáta a databázu objektov daného projektu, adresár sa kopíruje pri klonovaní z iného počítača
2. pracovný adresár (working directory) – obsahuje lokálnu kópiu jednej verzie projektu, tieto súbory sú z komprimovanej databázy v adresári Git
3. oblasť pripravených zmien (staging area) – je jednoduchý súbor, väčšinou uložený v adresári Git. Obsahuje informácie o budúcej revízii. Niekedy je označovaný aj ako index



### Práca v Git-e

1. Zmením súbory v pracovnom adresári
2. Súbory pripravím k uloženiu tak, že vložím ich snímky do stage area
3. Zapíšem revíziu. Snímky zo stage area sa zapíšu trvalo do adresára Git

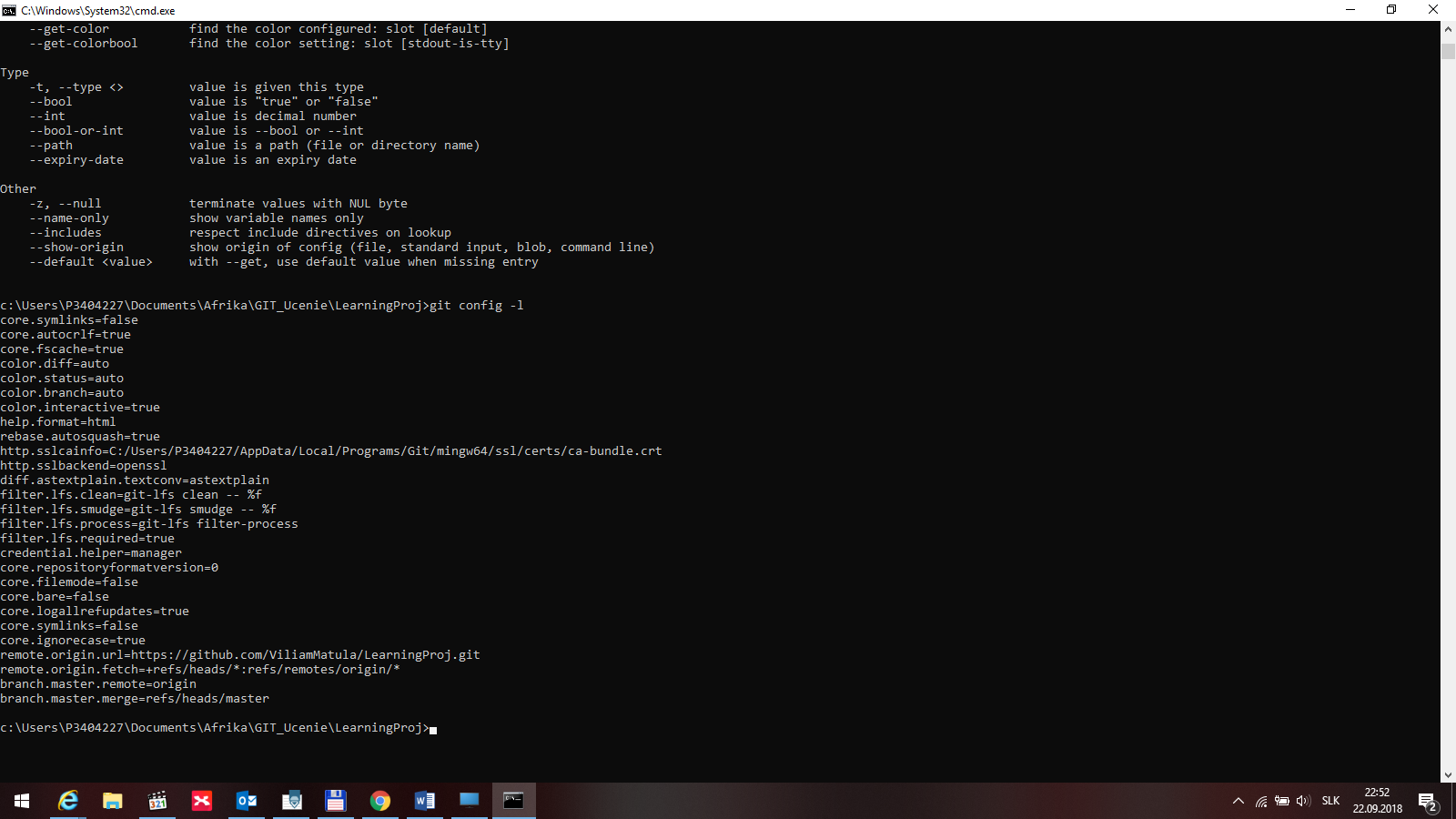
* ak sa nachádza konkrétna verzia súboru v adresári Git, je považovaná za zapísanú
* pokiaľ je modifikovaná verzia pridaná do stage area je považovaná za pripravenú k zápisu
* ak bola od poslednej operácie checkout zmenená, ale nebula pripravená k zápisu, je považovaná za zmenenú

### Nastavenia systému

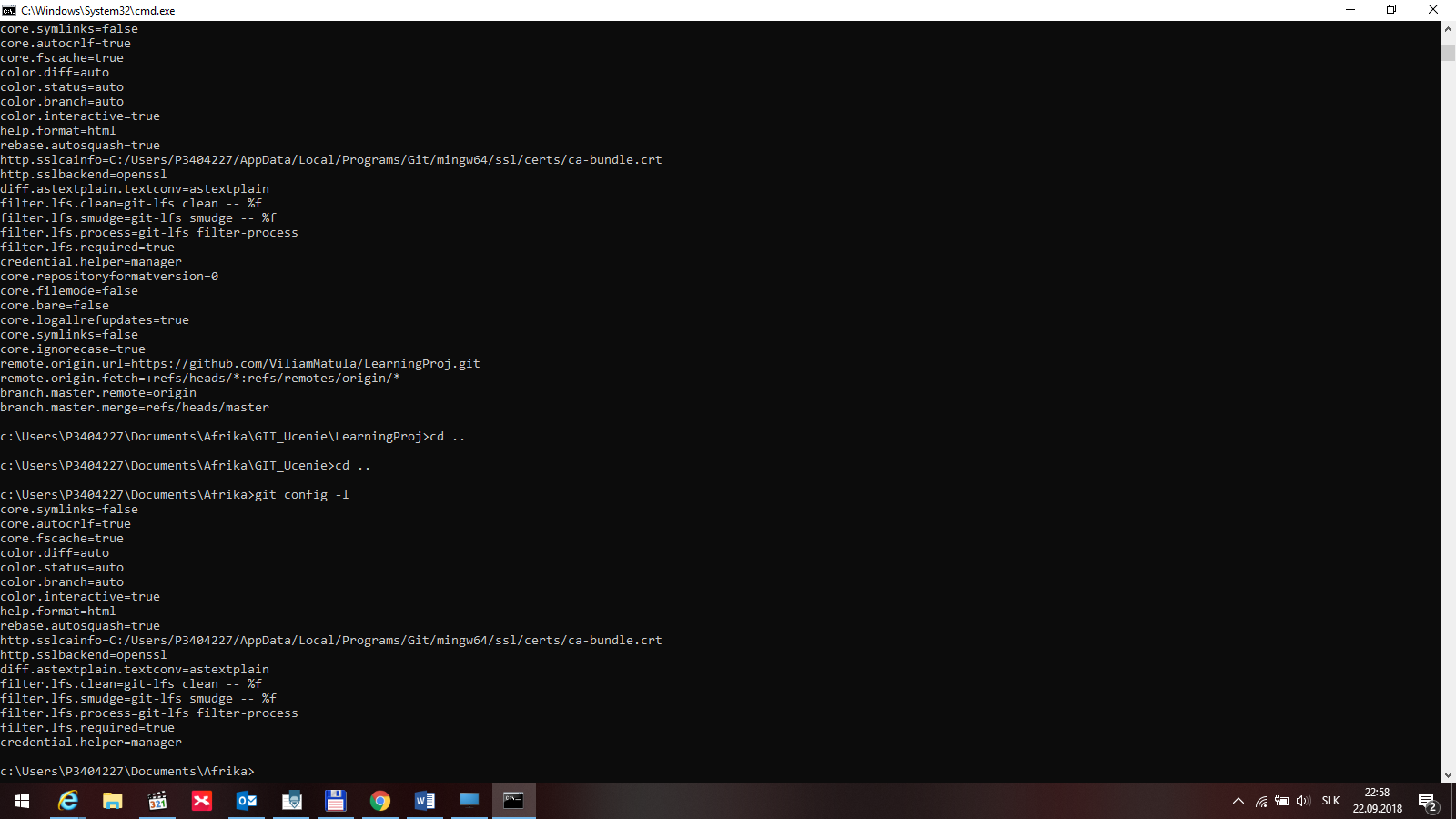
K nastaveniu používam príkaz

git config

Má rôzne parametre, napríklad keď použijem git config -l alebo git config --list vypíšu sa všetky nastavenia:

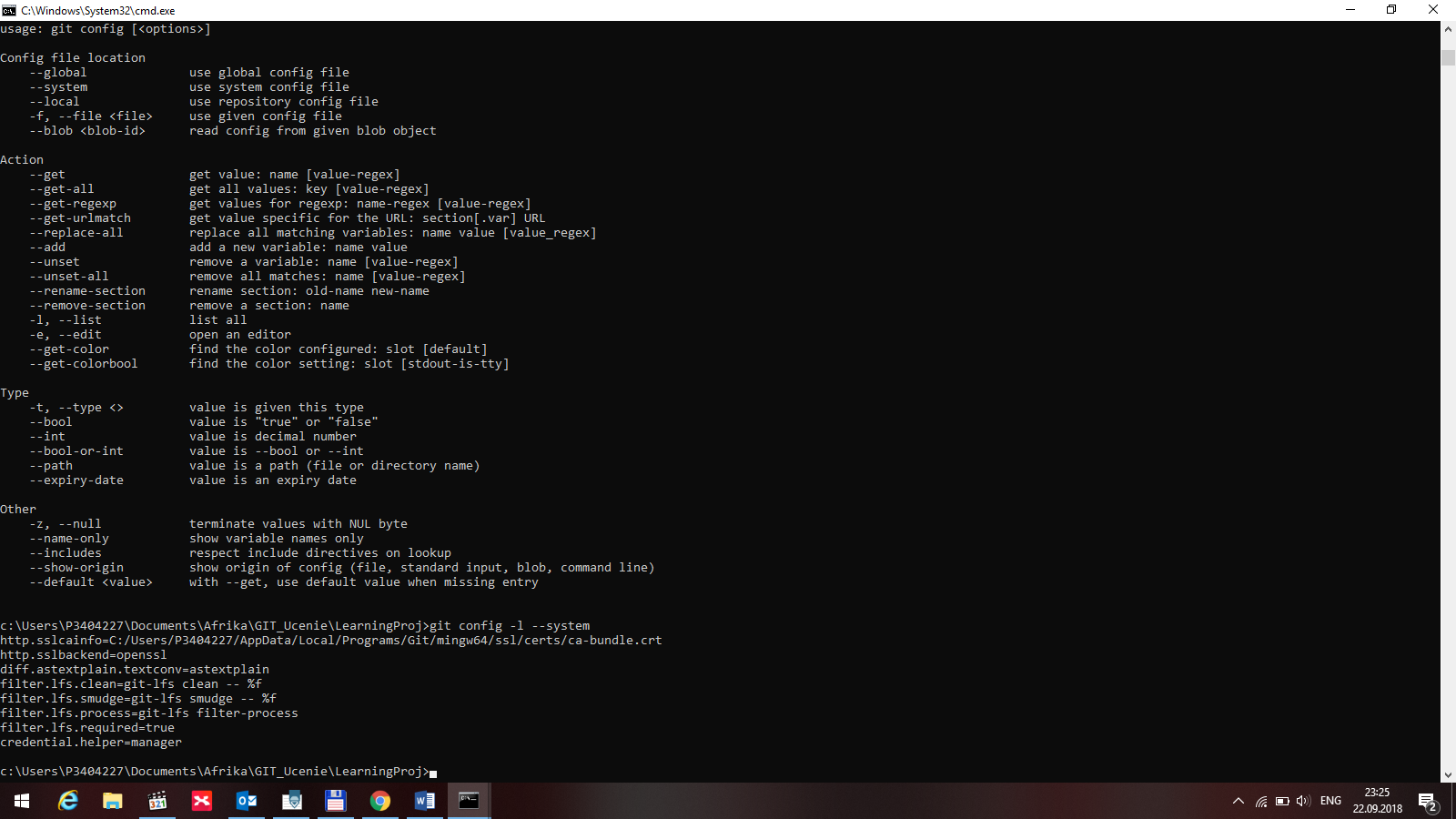


Keď spustím príkaz vo foldri pracovnej kópie, vypíšu sa aj informácie o danej repository, keď spustím mio, sú tam len obecné Git-ovské veci…

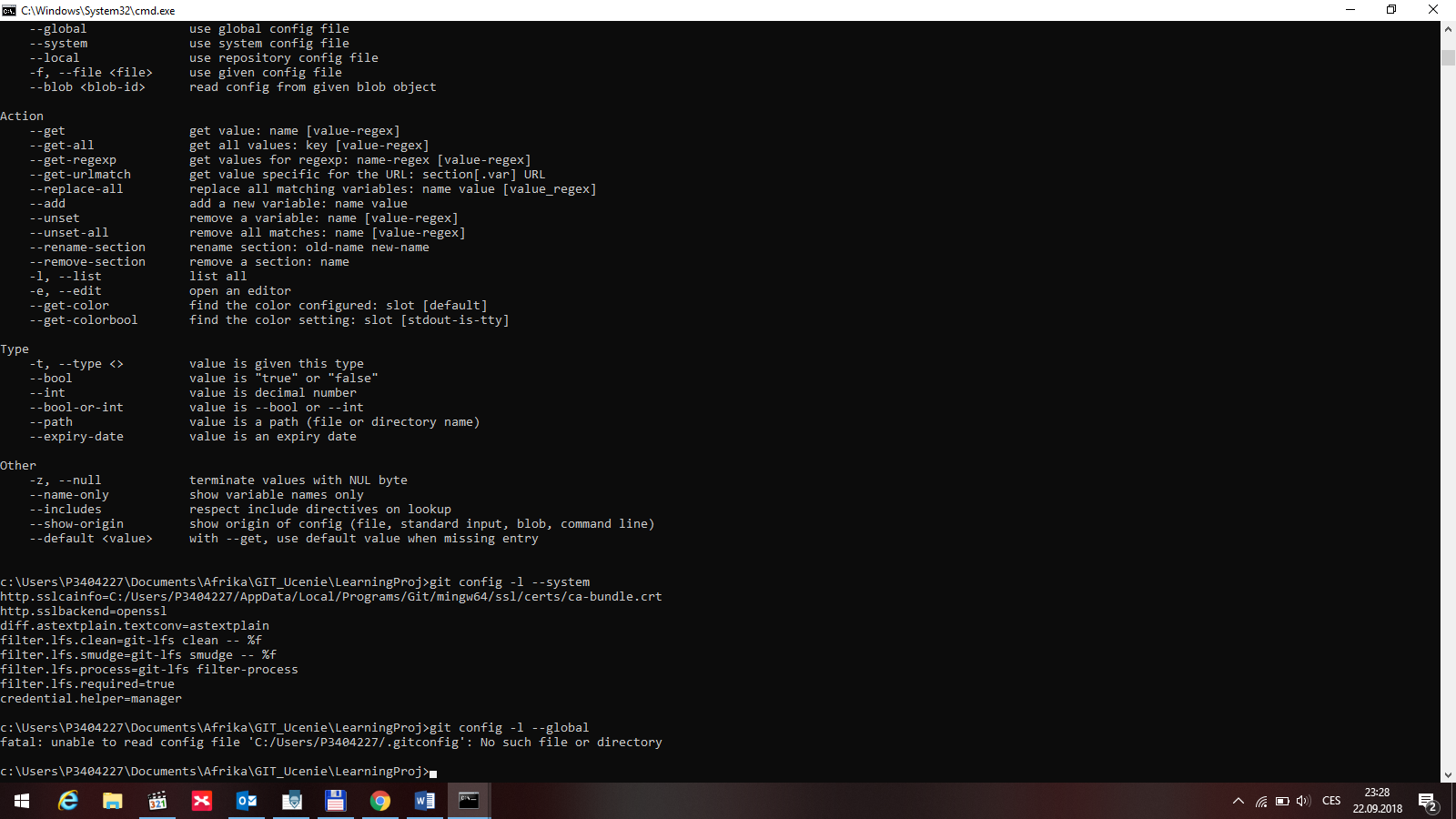


Konfiguračné súbory sú troch druhov:

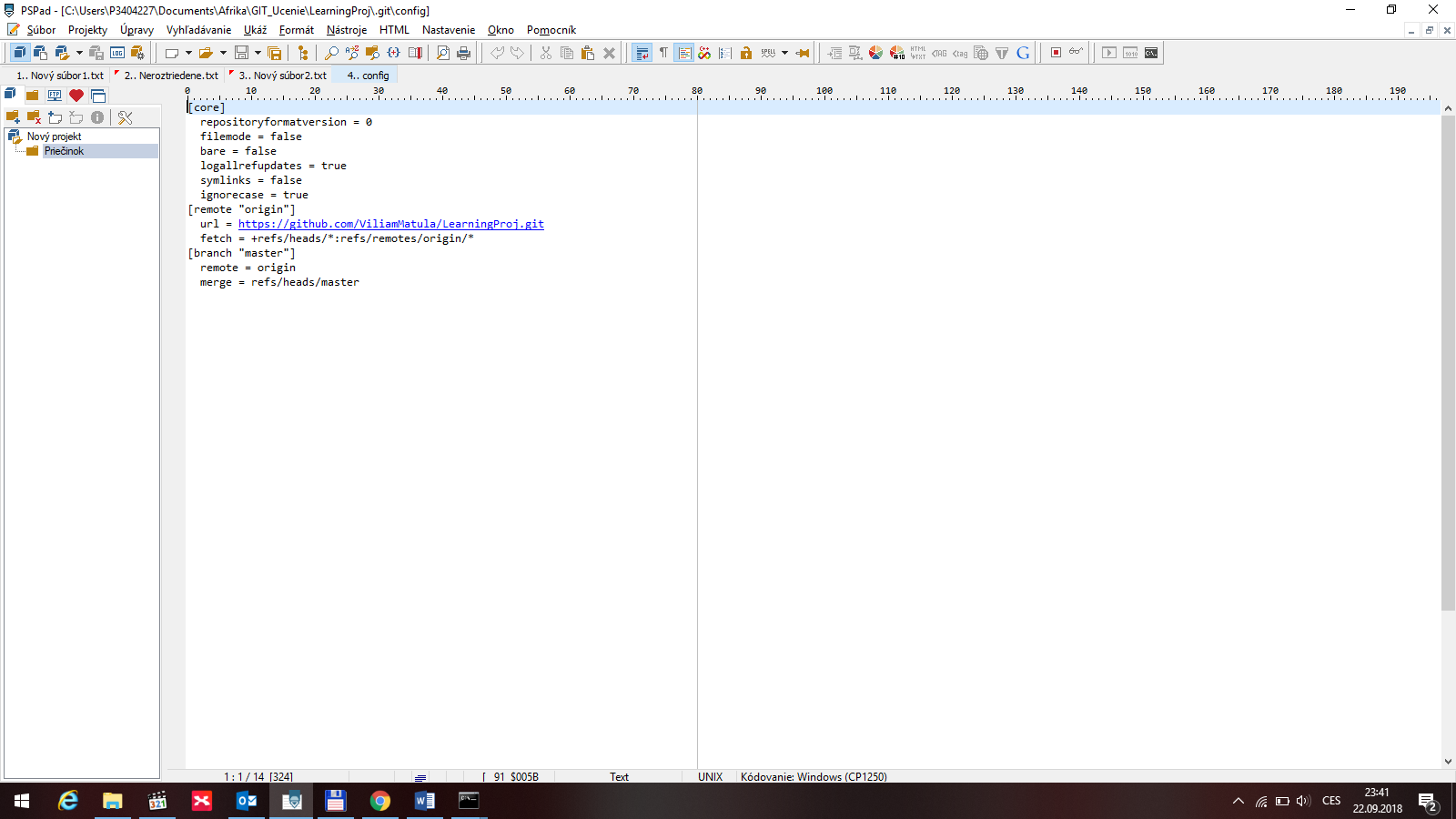
1. systémový cfg súbor - platí pre všetkých užívateľov počítača: git config -l --system



1. cfg viazaný na konkrétního užívateľa: git config -l --global [ja som ešte nemal nastavené]



1. cfg súbor daného repository vo foldri .git sa nachádza súbor config



Dôležité je nastaviť užívateľa:

git config --global user.name "Viliam Matula"

git config --global user.email viliam.matula@ness.com

Ak nepoužijem parameter global nastavím identitu pre aktuálnu repository, kde sa práve nachádzam

Keď chcem zobraziť nastavenie nejakého parametra:

git config user.name

### Nápoveda v Git-e

git help <příkaz>

git <příkaz> --help

**Vytvorenie novej repository**

git init

**Ďalšia práca s repository**

Pridanie sledovania súborov v rámci repository

git add \*.c

git add README

Podobne ako v Mercuriali mám súbor .gitignore v ktorom môžem vydefinovať súbory, ktoré má git ignorovať. Keď zmením nejaké súbory, znovu zadám git add - to nie je len na nové súbory, je to aj na poznačenie, že modifikovaný súbor dávam do stage area a je pripravený na commit. Ak by som upravil nejaký existujúci súbor a zadal by som git commit, nič by sa nezapísalo, lebo som nepripravil zmenený súbor na commit pomocou git add. Aby som mohol robiť bez toľkých príkazov, môžem robiť commit s prepinačom –a.

Keď mám zmeny hotove, urobím commit

git commit -m "initial project version"

Môžem si vyklonovať existujúce repository. Pokiaľ chcem premenovať, uvediem aj nové meno:

git clone git://github.com/schacon/grit.git mygrit

Keď mám pracovnú kópiu súborov a ešte som nezačal robiť žiadne zmeny mám ju **checkout**-ovanú.Následne by som chcel urobiť niekolko zmien a zaznamenal v bode, ktorý mi príde dôležitý - urobím **commit**.

Každý súbor v pracovnom adresári môže byť v dvoch stavoch:

1. nesledovaný (untracked)
2. sledovaný (track) – tie ešte môzem rozdeliť
   1. zmenený (modified)
   2. nezmenený (unmodified)
   3. pripravený k zapísaniu (staged)

Životný cyklus súborov:



Kontrola stavu súborov v repository: git status

Môžem použiť aj skrátenú verziu: git status –s

Ak by som chcel podrobnejšie vidieť aj čo sa konkrétne v súboroch zmenilo: git diff

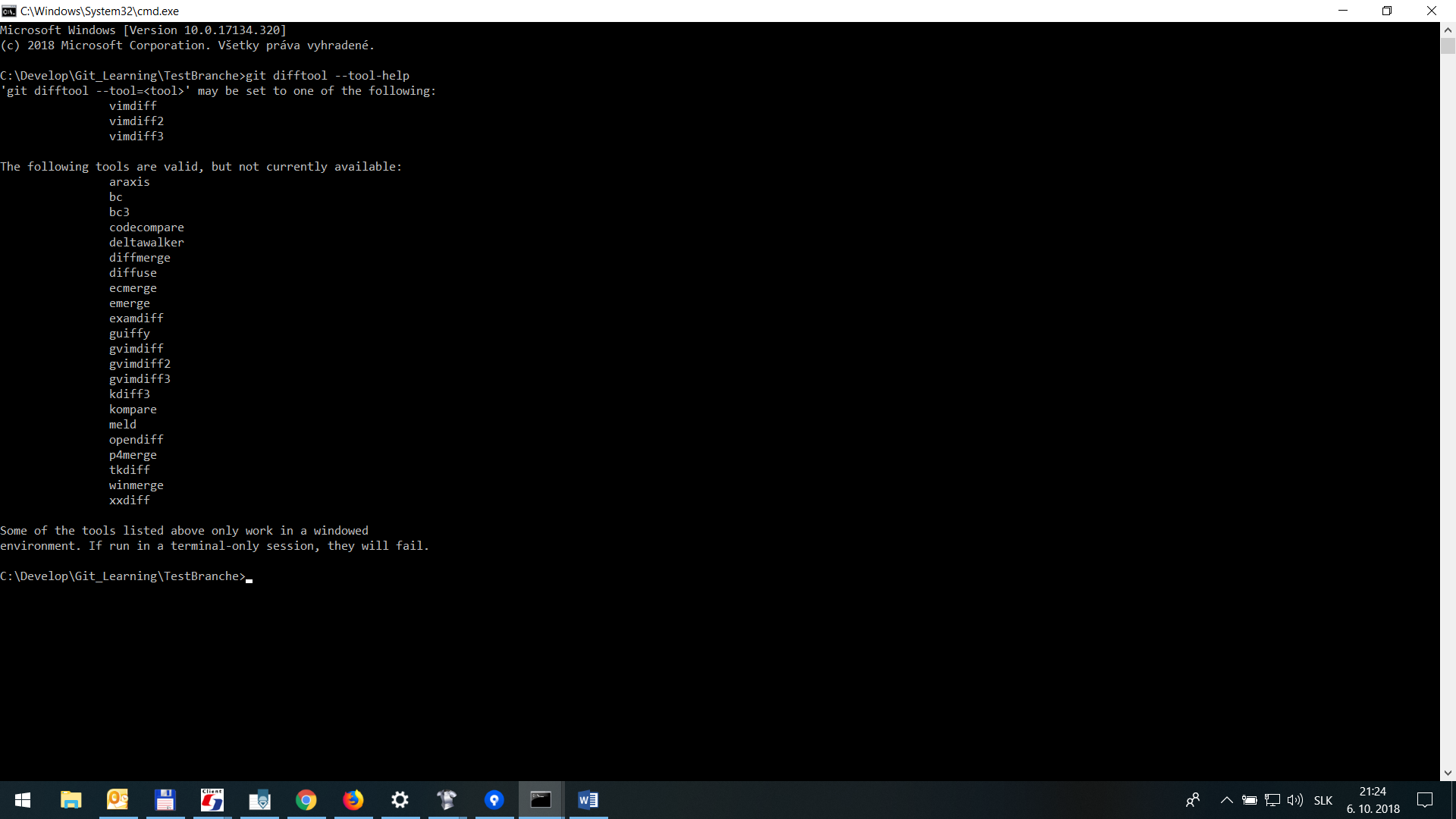
Tento príkaz porovná čo mám v oblasti pripravovaných zmien s obsahom pracovného adresára. Výsledkom je, aké zmeny som urobil a ešte nepripravil ku commitu. Ak chcem pozrieť čo už som poznačil jako pripravené ku commitu v porovnaní s predošlým commit-om: git diff –staged

rovnaký význam má: git diff –cached

K zobrazeniu rozdielov môžem použiť rôzne nástroje. Príkazom:

git difftool --tool-help

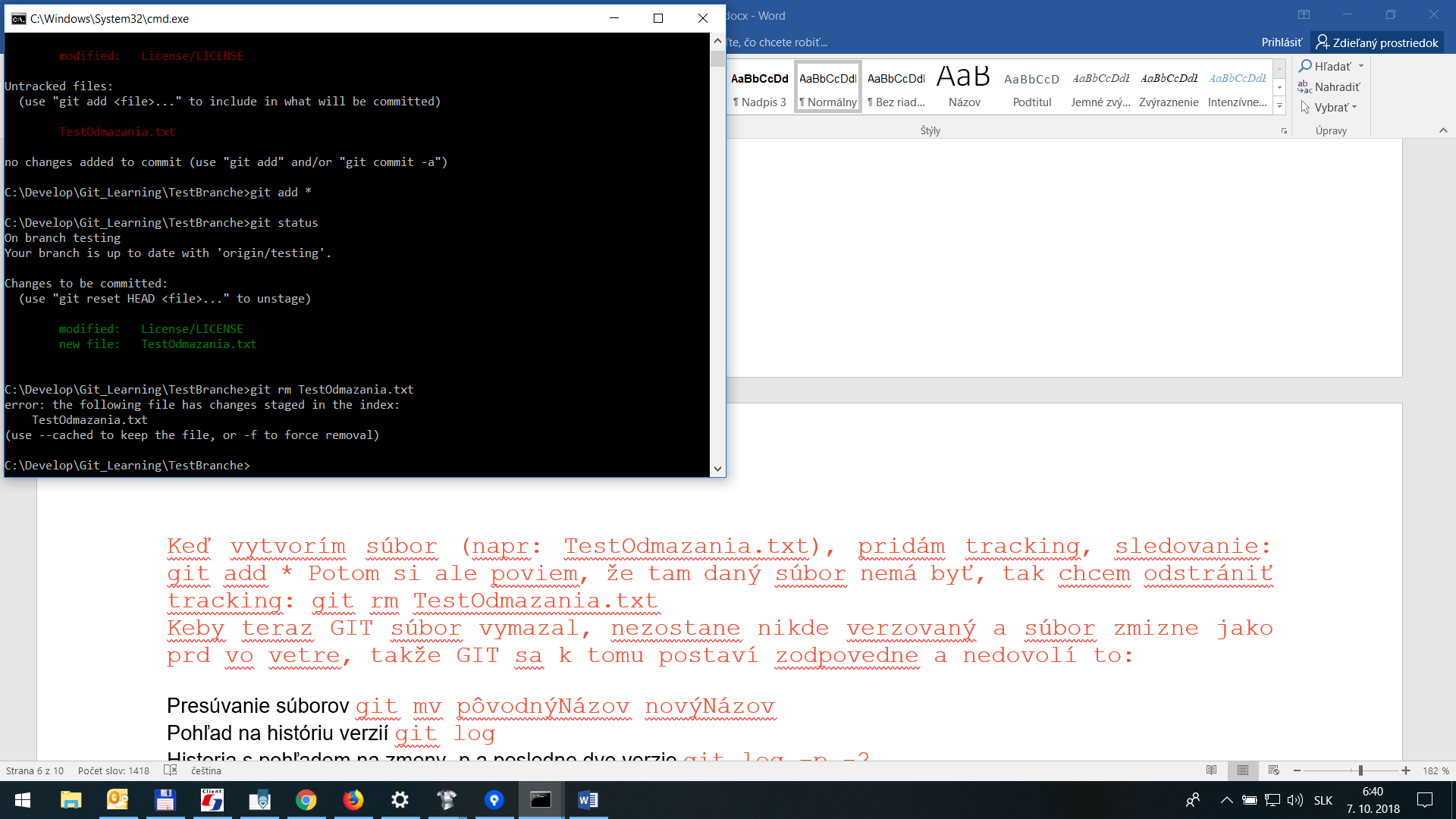
si zobrazím dostupné



Odobrať sledovanie súboru v repository git rm

Keď vytvorím súbor (napr: TestOdmazania.txt), pridám tracking, sledovanie: git add \* Potom si ale poviem, že tam daný súbor nemá byť, tak chcem odstrániť tracking: git rm TestOdmazania.txt

Keby teraz GIT súbor vymazal, nezostane nikde verzovaný a súbor zmizne jako prd vo vetre, takže GIT sa k tomu postaví zodpovedne a nedovolí to:



Na výber mám že použijem git rm TestOdmazania.txt –f prepínač force, ktorý jednoducho súbor odstráni zo sledovania a zároveň vymaže z disku. Alebo môžem použiť git rm TestOdmazania.txt –cached kedy sa odstráni tracking ale súbor zostane na disku.

Presúvanie súborov git mv pôvodnýNázov novýNázov

Pohľad na históriu verzií git log V defaultnom nastavení a bez ďalších parametrov vypíše revízie v obrátenom chronologickom poradí, takže posledné úpravy na začiatku. Príkaz mi vypíše všetky commity so svojím kontrólnym súčtom SHA-1, menom autora commitu, emailom, dátumom a popisom commitu.

Voľba –p zobrazí prehľad zmien v danom commite diff –2 výpis posledných dvoch zmien – samozrejme môžem zadať akékoľvek číslo, akýkoľvek počet: git log -p -2

Ďalšie užitočné volby:

git log -–stat zobrazí pod každým commitom štatistiku – koľko súborov bolo zmenených (changed), koľko riadkov bolo do zmenených súborov vložené (insertions) a zmazaných (deletions). Na konci potom ešte súhrn informácií.

git log --pretty=oneline vo voľbe pretty mám možné hodnoty [oneline,short,full,fuller] s významami: výpis na jeden riadok,skrátený,plný a úplný. Ďalšia hodnota format môže pomôcť pri spracovaní výsledkov. Môžem dať výpisu presný format a potom presne viem čo mám vo výsledkoch očakávať. Jednotlivé značky formátu:

%H Hash revízie

%h Zkrácený hash revízie

%T Hash stromu (T ako tree)

%t Skrátený hash stromu

%P Hash rodičovských revízií (P ako parent)

%p Skratené hash rodičovských revízií

%an Meno autora (author name)

%ae E-mail autora (author e-mail)

%ad Dátum autora (formát rešpektuje --date=volba)

%ar Dátum autora, relatívny

%cn Meno autora revízie (committer name)

%ce E-mail autora revízie (committer e-mail)

%cd Dátum autora revízie (committer date)

%cr Dátum autora revízie, relatívne

%s Predmet (subject)

git log --pretty=format:"%h %s" --graph

Ďalší super parameter graph pomáha zobraziť vetvenie – je to síce graf vytvorený v textovom režime, ale lepšie jako drótom do oka.

--shortstat zobrazí len zmenené/vložené/zmazané z príkazu --stat.

--name-only za informáciami o revízii zobrazí zoznam zmenených súborov

--name-status zobrazí zoznam dotknutých súborov spolu s informáciou zmenené/vložené/zmazané

--abbrev-commit zobrazí len nniekoľko prvých znakov kontrólneho súčtu SHA-1 namiesto všetkých 40

--relative-date zobrazí dátum v relatívnom formáte (napr. „2 weeks ago“, tj. pred 2 týždňami) namiesto dátumu v úplnom tvare

--since, --after omezenie výpisu na revízie zapísané po zadanom dátume

--until, --before omezenie výpisu na revízie zapísané pred zadaným dátumom

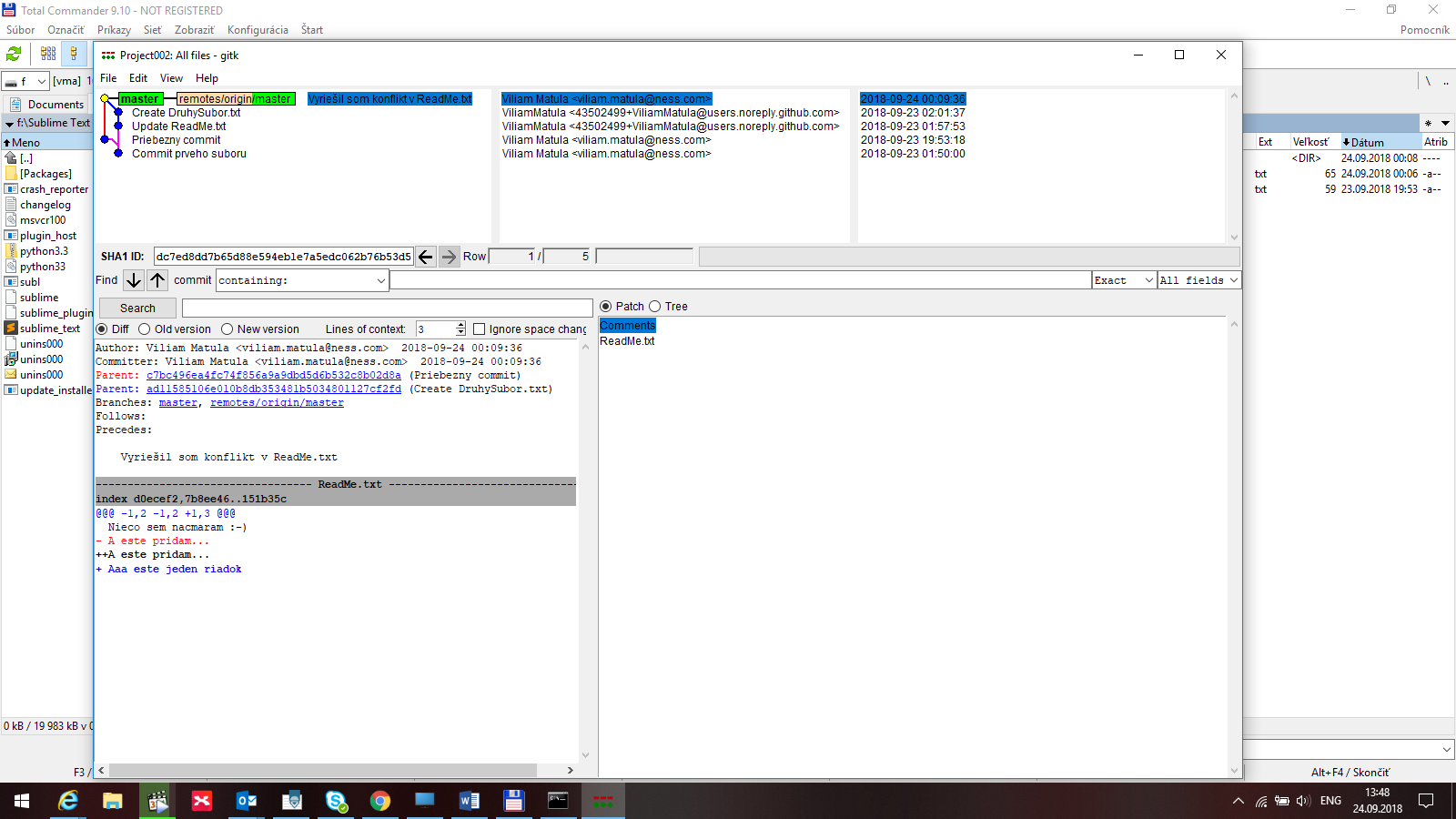
--author zobrazí len revízie, u ktorých autor príspevku sa rovná zadanému reťazcu

--committer zobrazí len revízie, u ktorých autor commitu sa rovná zadanému reťazcu

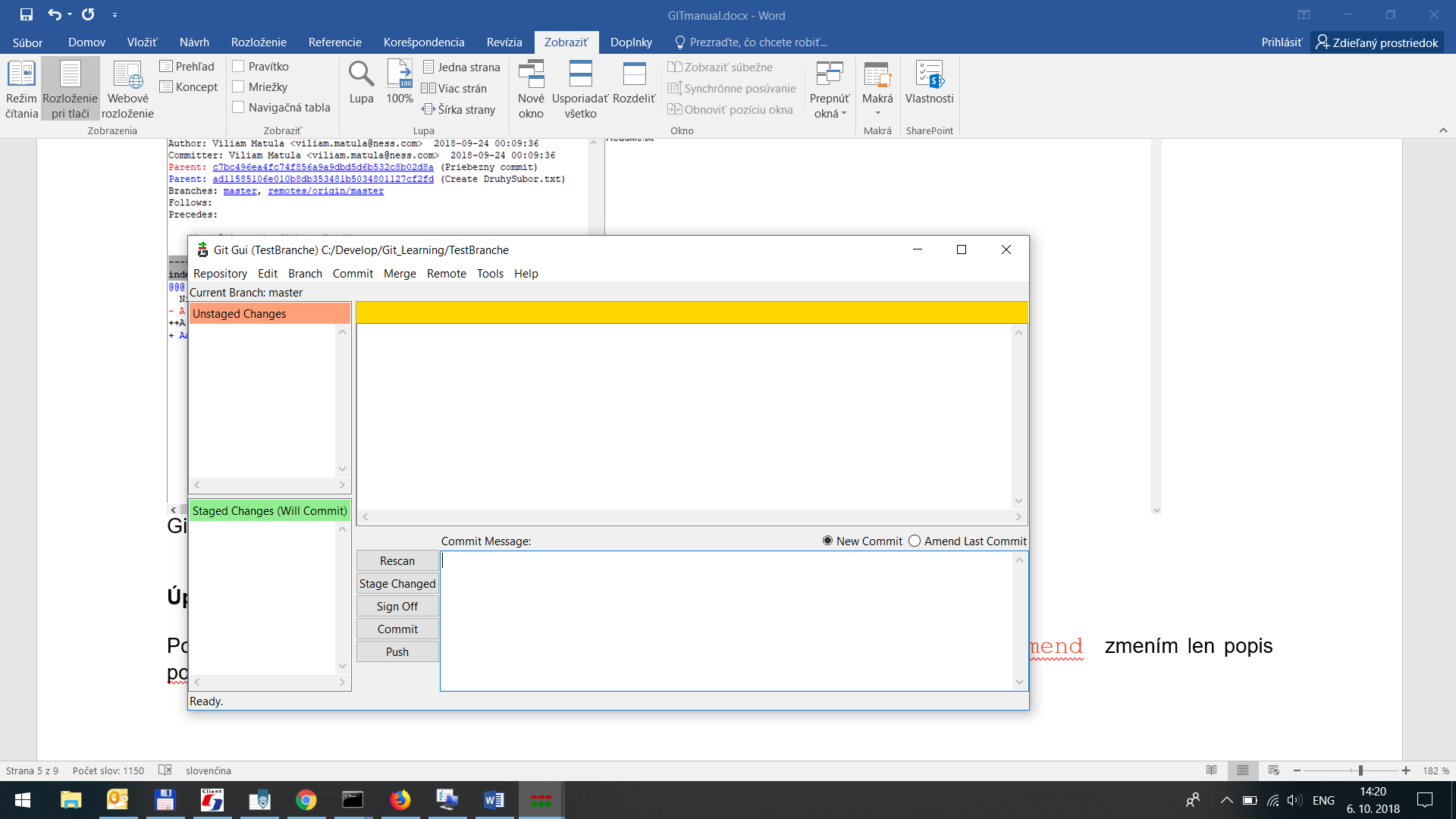
--grep zobrazí len revízie, u ktorých sa v popise commitu nachádza zadaný reťazec

-S zobrazí len revízie, ktoré pridali alebo odobrali zadaný reťazec - hľadá sa v zmenených súboroch

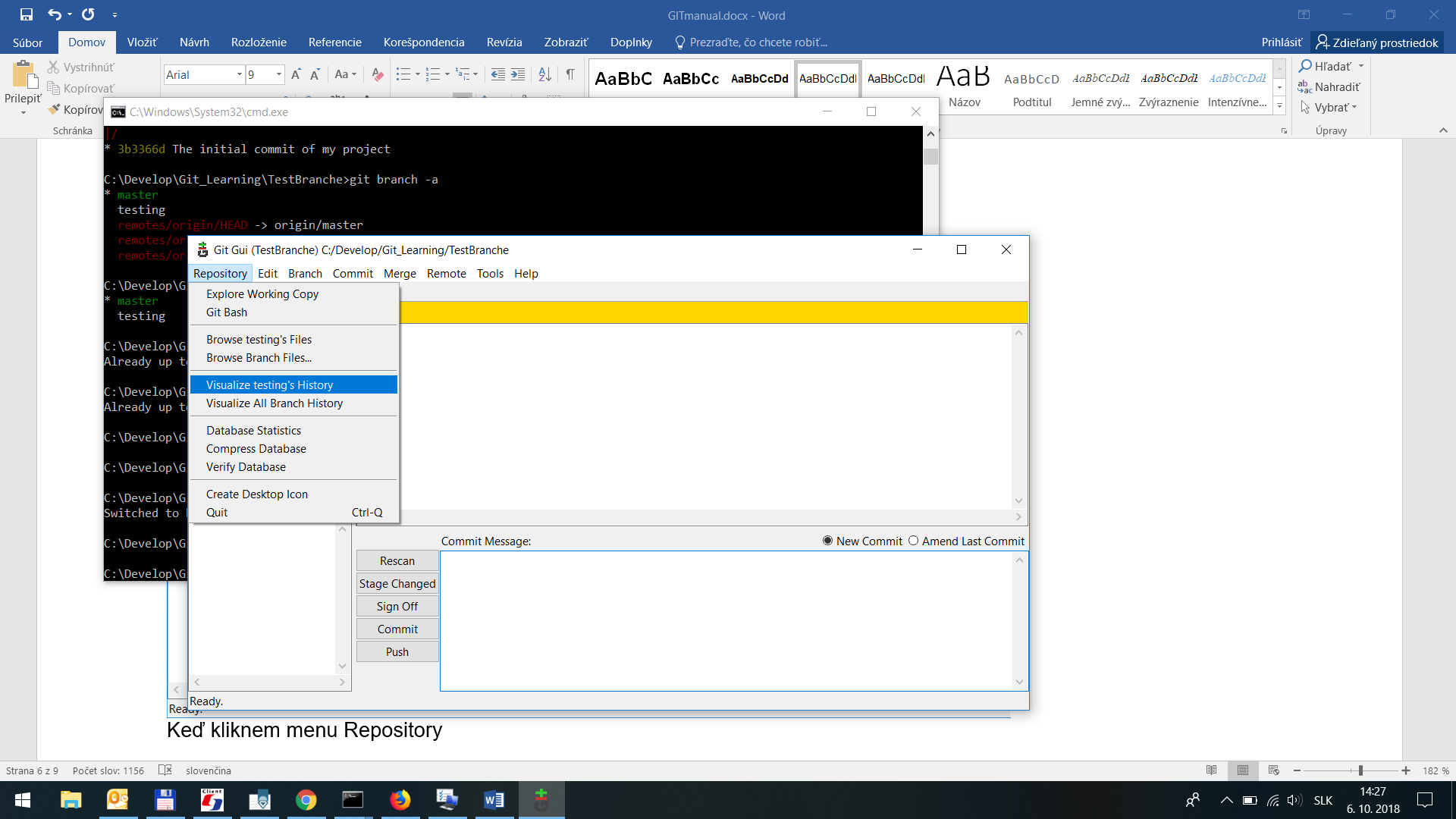
Grafická obdoba logu: gitk



Alebo použijem git gui



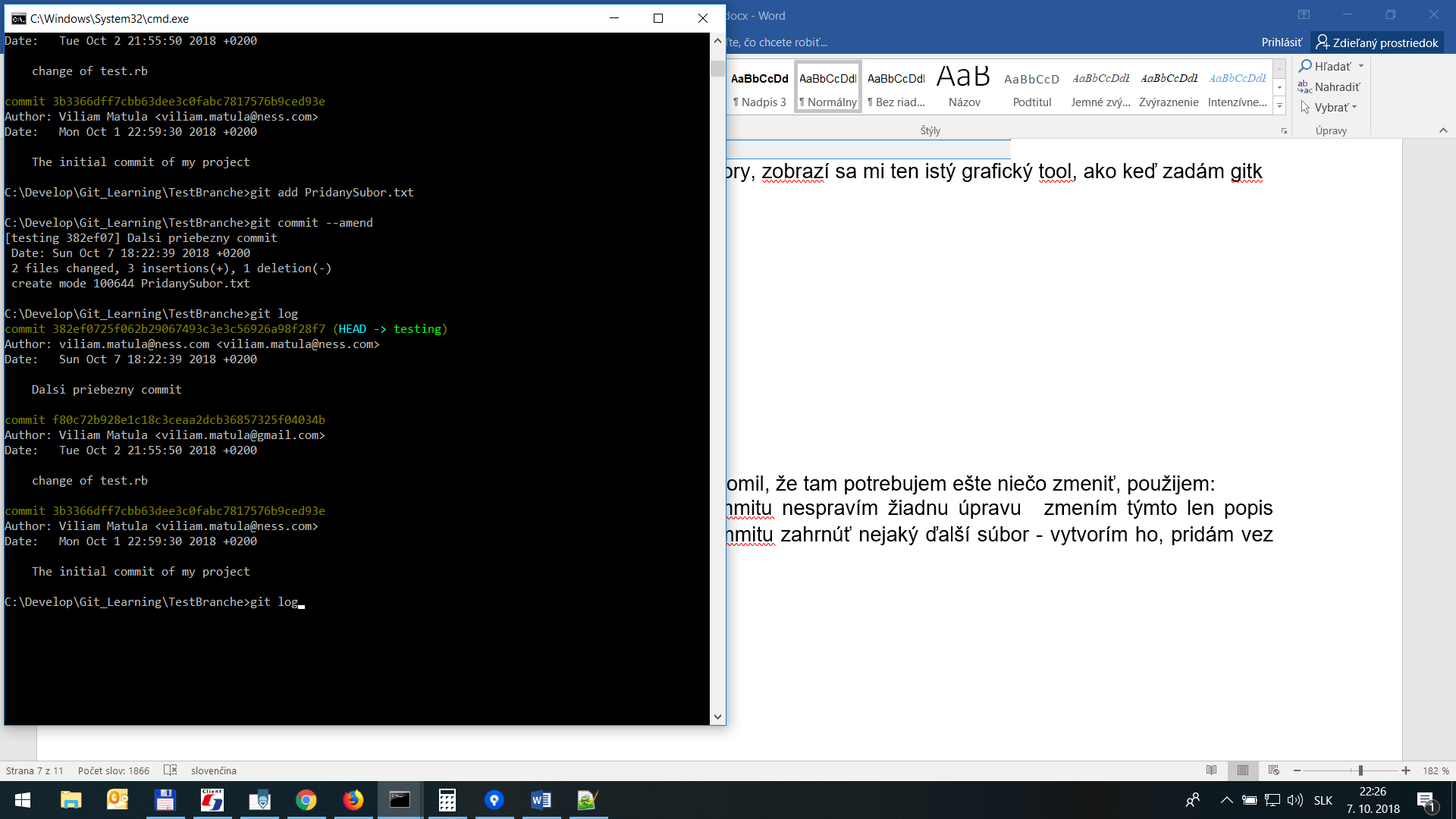
Keď kliknem menu Repository\Visualize [názov brenche] history, zobrazí sa mi ten istý grafický tool, ako keď zadám gitk



### Návrat k predošlému stavu

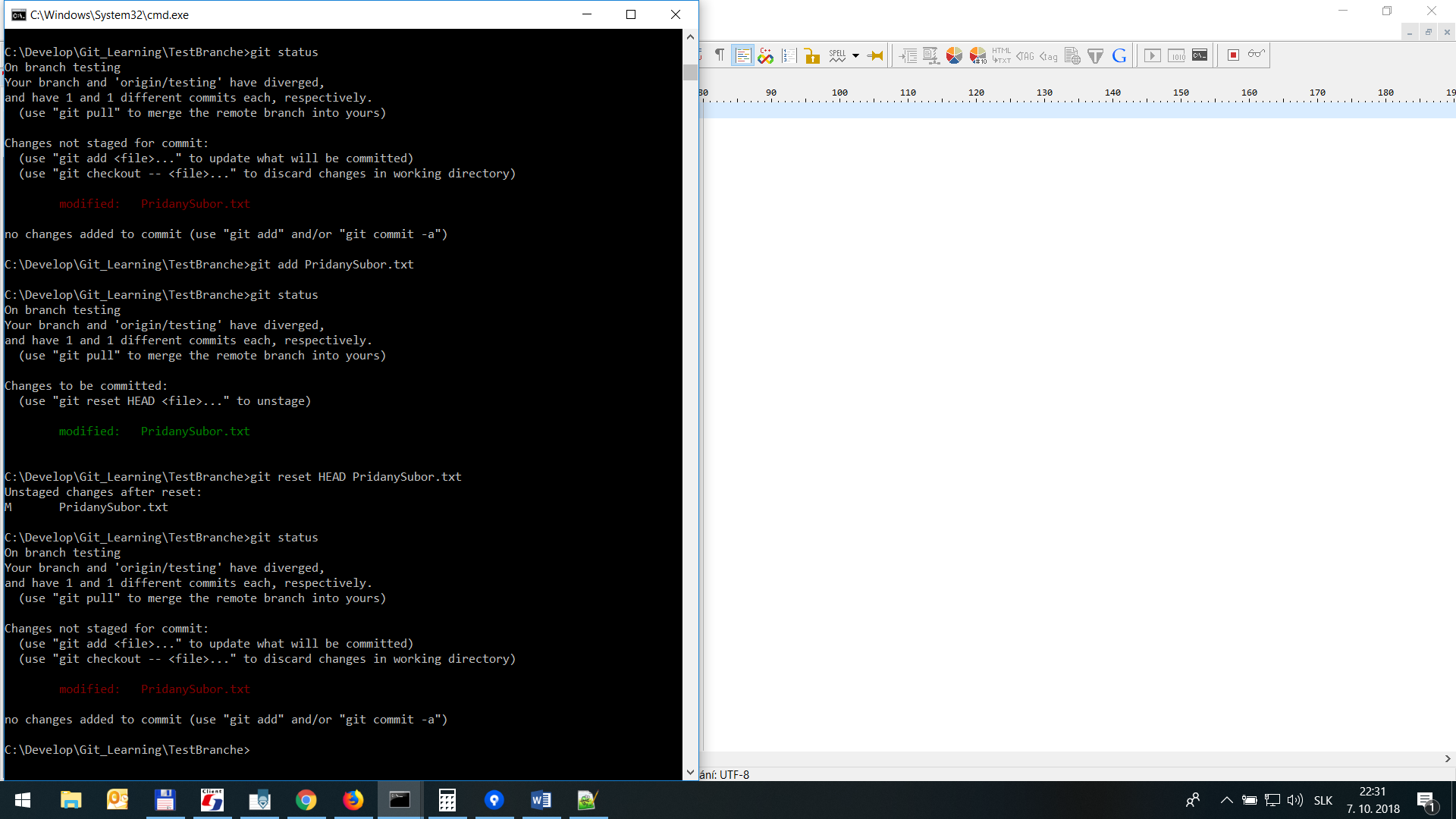
Pokiaľ som príliš skoro urobil commit a následne som si uvedomil, že tam potrebujem ešte niečo zmeniť, použijem:

git commit –-amend ak som od posledného commitu nespravím žiadnu úpravu zmením týmto len popis posledneho commitu. Ak ale chcem ešte do posledného commitu zahrnúť nejaký ďalší súbor - vytvorím ho, pridám vez add a použijem amend:



Ďalšia situácia, keď mám upravený súbor, pomocou add to pripravím do stage area a zistím, že ho ešte potrebujem upraviť, použijem:

git reset HEAD PridanySubor.txt

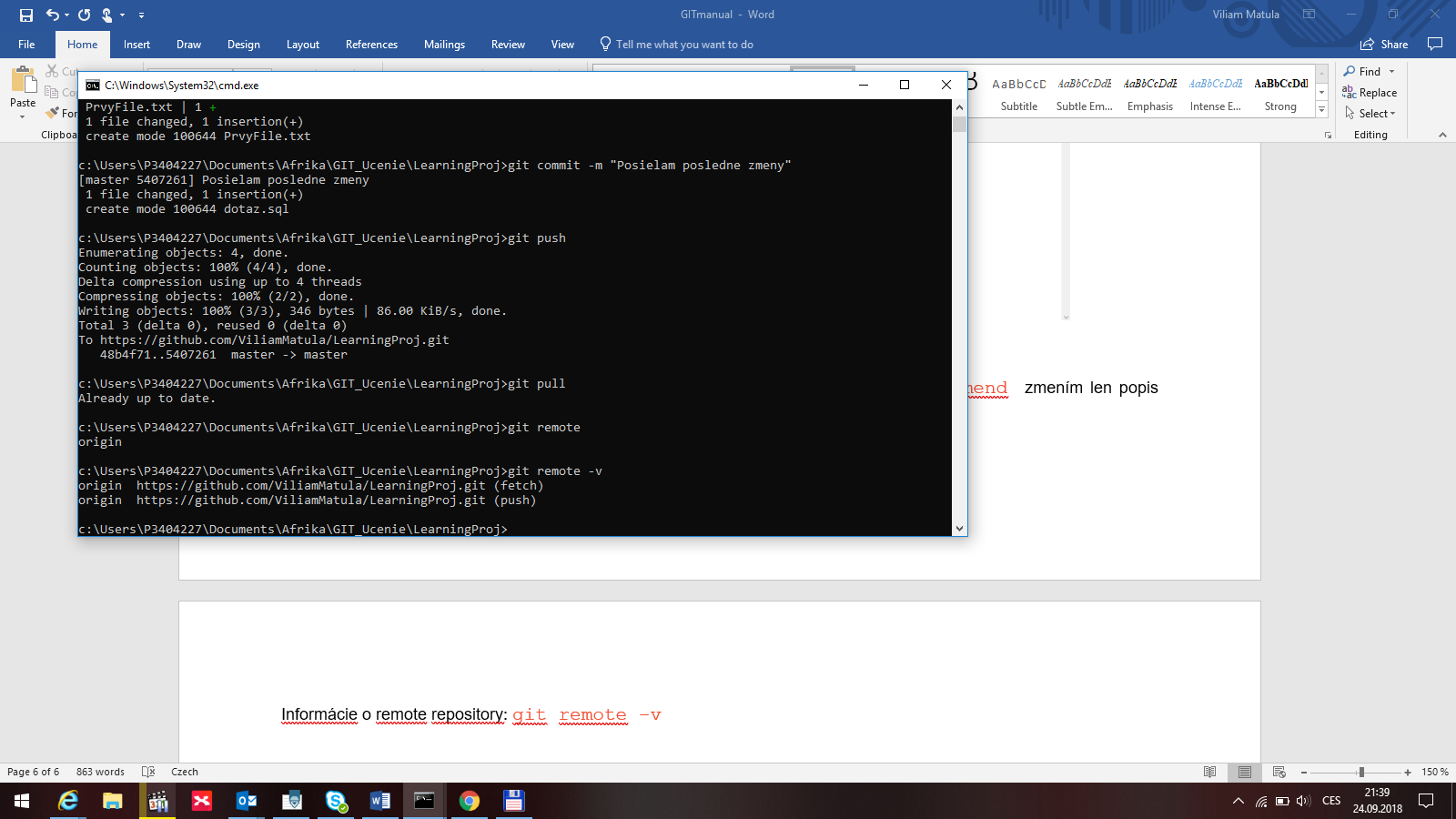


Ak chcem zahodiť úpravy, ktoré som spravil v zmenenom súbore:

git checkout -- PridanySubor.txt

### Správa vzdialeného repository

Informácie o remote repository: git remote -v prepínač v zobrazí url vzdialenej repository



Pridanie remote repository git remote add [zkrácený název] [url] do zoznamu remote repository, ktoré máme nadefinované

git remote add origin https://github.com/ViliamMatula/TestRemoteRepo.git

Stiahnutie zmien zo zdialeného repository git fetch [název vzdáleného repozitáře]

Keď mám repository vyklonovanú zo vzdialeneho, možem zadávať git fetch origin

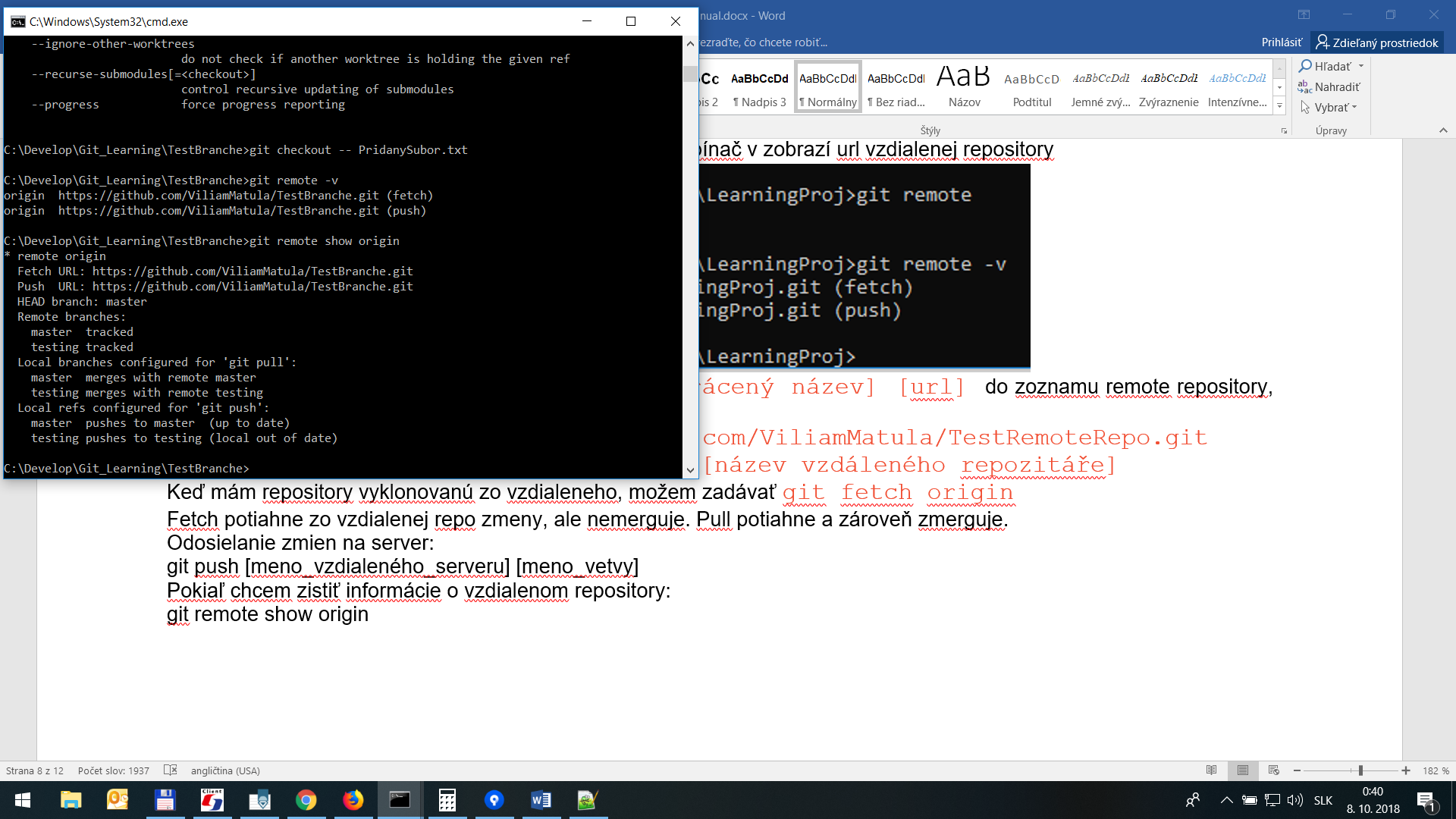
Fetch potiahne zo vzdialenej repo zmeny, ale nemerguje. Pull potiahne a zároveň zmerguje.

Odosielanie zmien na server:

git push [meno\_vzdialeného\_serveru] [meno\_vetvy]

Pokiaľ chcem zistiť informácie o vzdialenom repository:

git remote show origin



Premenovanie referencie na remote repository: git remote rename pb paul

Odstránenie referencie na remote repository: git remote rm paul

**Práca so značkami**

Výpis značiek v repository: git tag

Obmedzenie výpisu podľa zadaného reťazca: git tag -l "v1.8.5\*"

Rozlišujem dva druhy značiek - prosté (lightweight) a anotované (annotated). Prosté sú obyčajné ukazatele na konkrétnu revíziu. Anotované sú GITom ukladané ako plnohodnotné objekty. Obsahujú:

* autora značky - tagger
* email a dátum
* správa značky - tagging message
* môžu byť podpísané - signed a overené - verified - v programe GNU Privacy Guard GPG

Anotovanú vytvorím pomocou parametra -a git tag -a v1.4 -m "my version 1.4"

Prostá značka sa zadáva bez parametrov: git tag v1.4-lw

Keď pushujem, značky sa automaticky neprenášajú, musím zadať: git push origin [názov-značky]

Alebo môžem poslať všetky znacky, ktoré ešte nie sú v remote repository: git push origin --tags

Checkoutom sa nemôžem odvolávať na značku! Môžem so značkou vytvoriť branch so značkou a na ňu sa potom odvolať:

git checkout -b [jméno-větve] [jméno-značky]

Prepnem na práve vytvorenú novú branch: git checkout -b version2 v2.0.0

Pozor ale - značka zostane na danej revízii, ale keď pridám nové commity, branch sa posunie vždy s čelom branche...

**Práca s aliasmi**

Môžem si nadefinovať aliasy - vlastné príkazy:

git config --global alias.unstage "reset HEAD --"

keď to takto nadefinujem, tak môzem namiesto: git reset HEAD -- fileA

zadať: git unstage fileA

alebo

git config --global alias.last "log -1 HEAD"

Potom pre zobrazenie poslednej revízie stačí zadať: git last

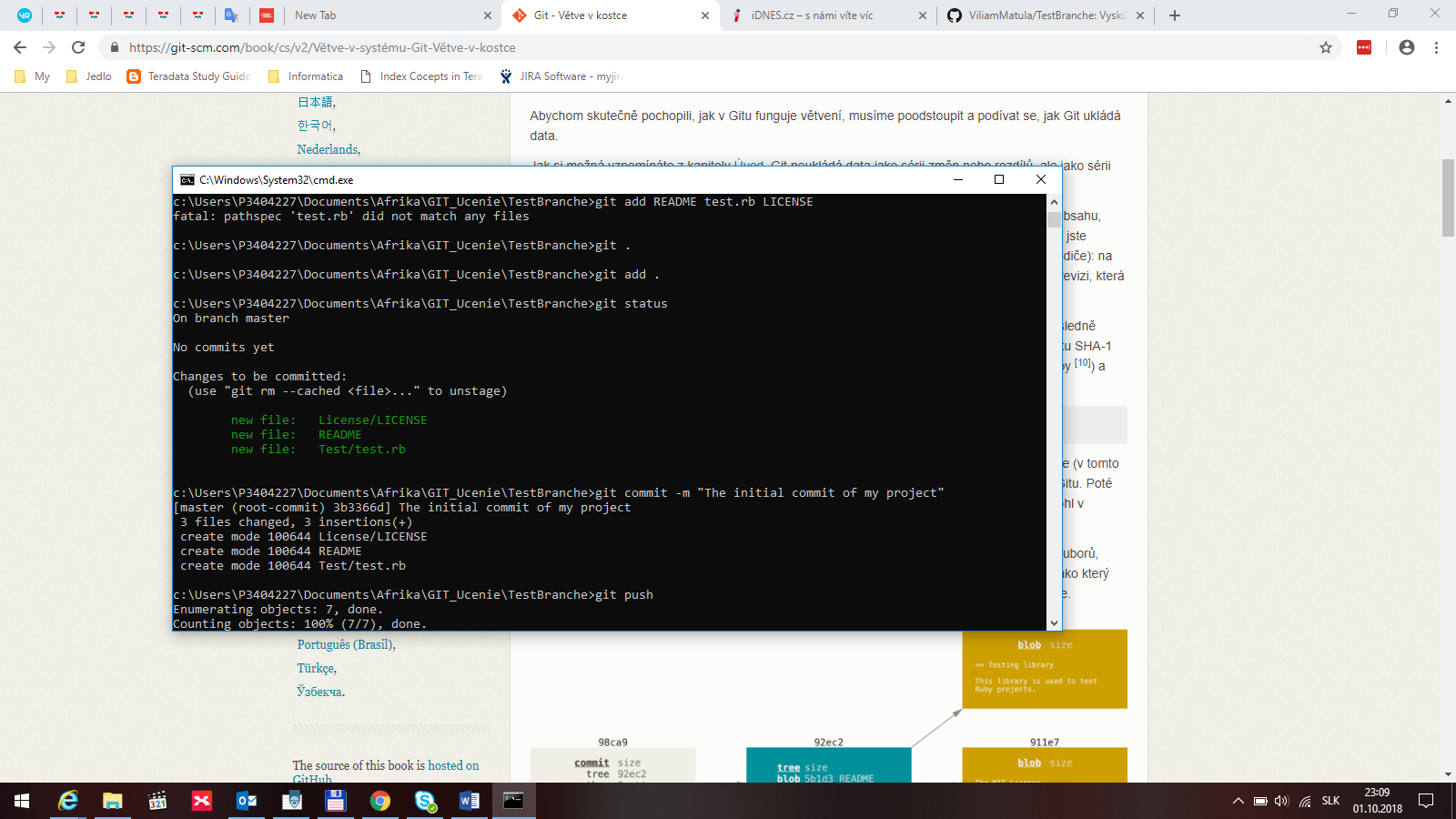
Ak chcem zadať externý príkaz, musím na začiatku zadať !

napr: git config --global alias.visual "!gitk"

potom volám: git visual a zavolá sa externý nástroj gitk

### Práca s vetvami

Vytvorím si novú repository, aby som si vyskúšal prácu s vetvami.



Vytvoril som tri súbory, pridal som ich do repository a zakomitoval som zmeny.

Git adresár obsahuje v tomto bode päť objektov – tri objekty troch súborov, objekt repository, objekt commitu:



Keď vytváram ďalšie úpravy, commit vždy ukazuje na predchodcu:



Ako vidieť neukladá sa delta, ale vždy odkazujem na posledný snímok (snapshot) daného súboru. Vetva v GITe je len ľahko presunuteľný ukazateľ na niektorú z revízií. Po vytvorení git init vzniká implicitná branche s názvom master.

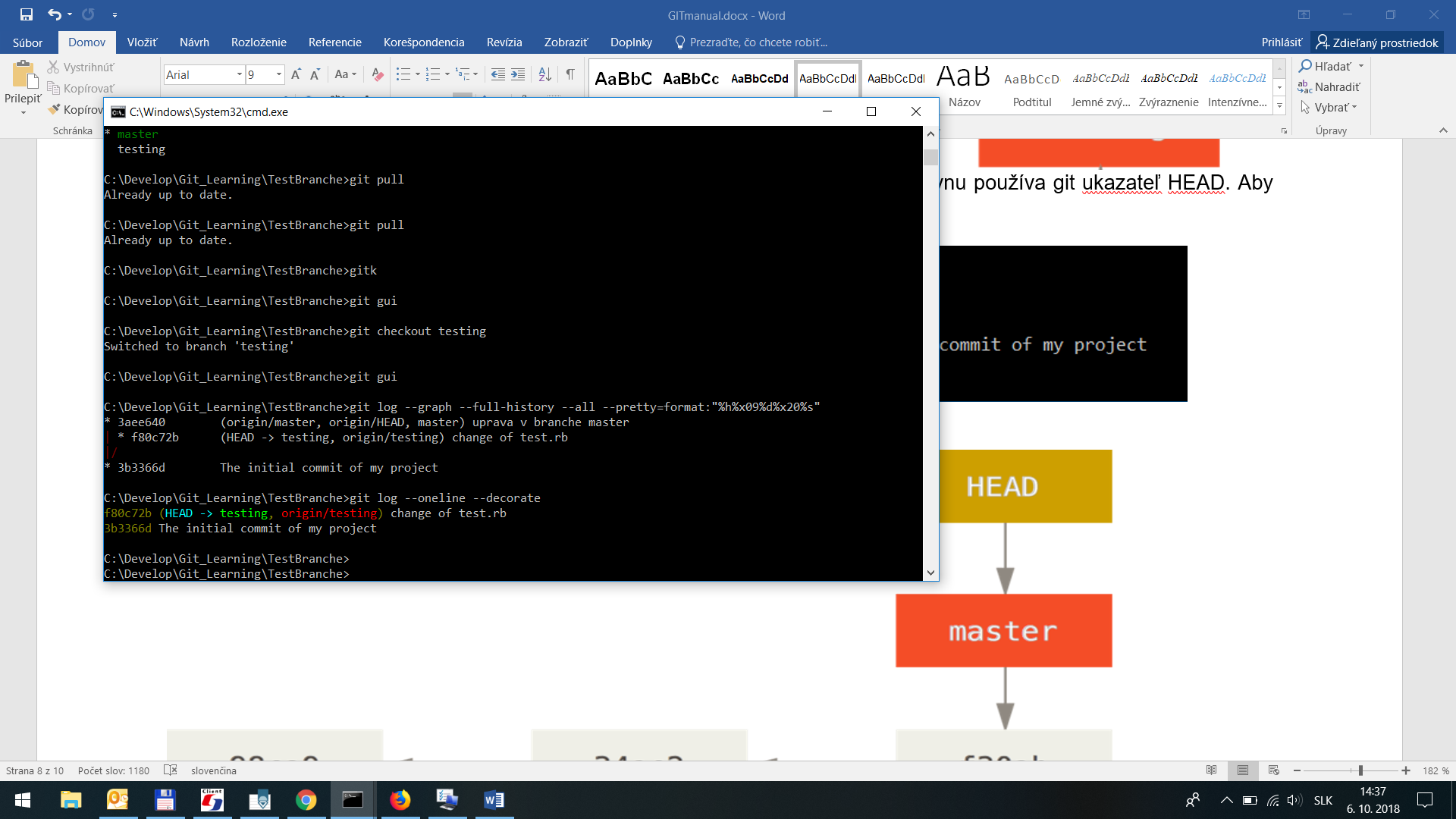
Kedykoľvek ju môžem premenovať. Keď vytvorím novú verziu vetva sa automaticky posunie na novú verziu. Keď vytvorím novu branche:

git branch testing

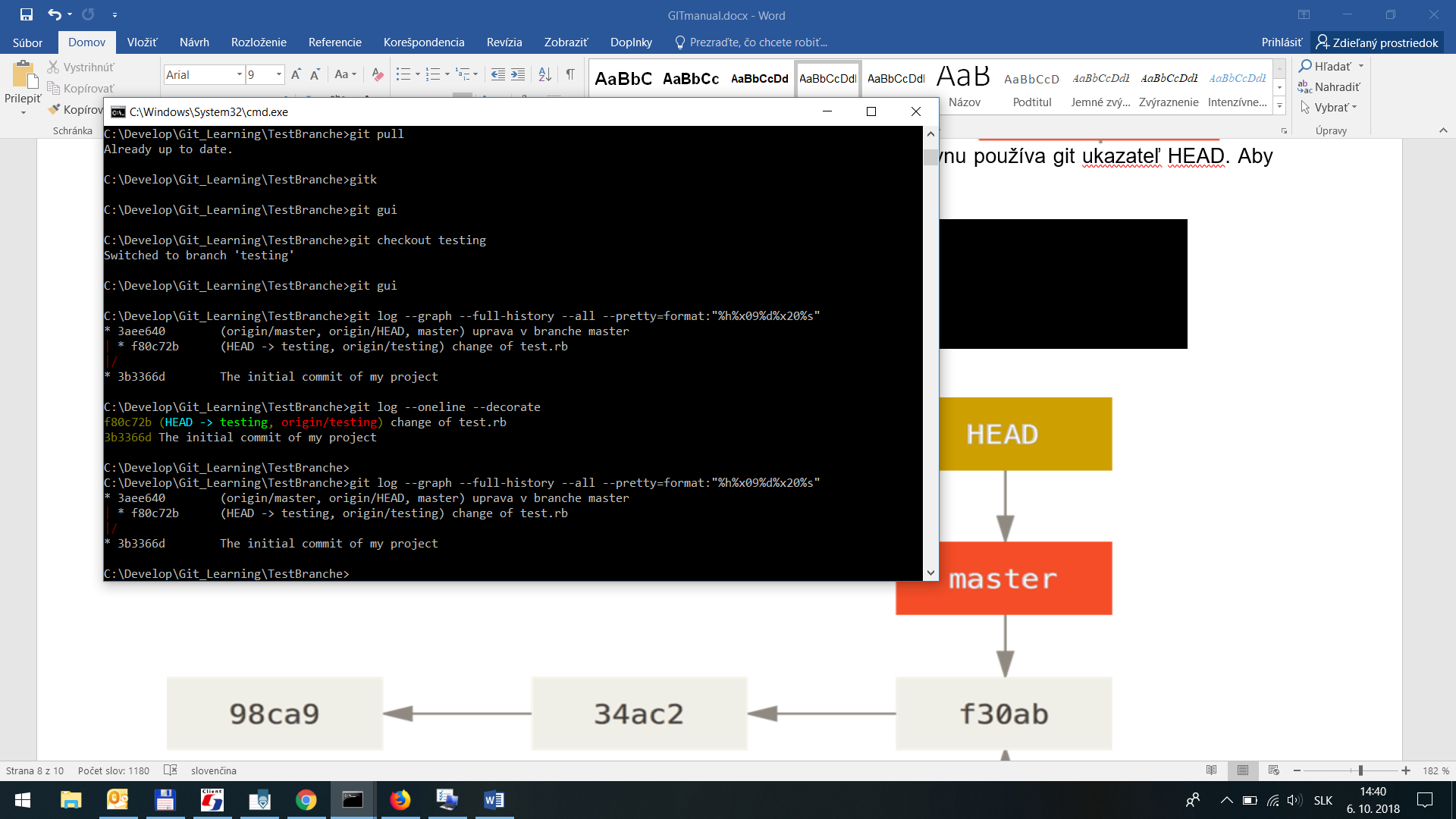


Na jednej revíii mám teraz dve branche, aby som spoznal ktorú branche mám aktívnu používa git ukazateľ HEAD. Aby som videal aj head, použijem:

git log --oneline –-decorate



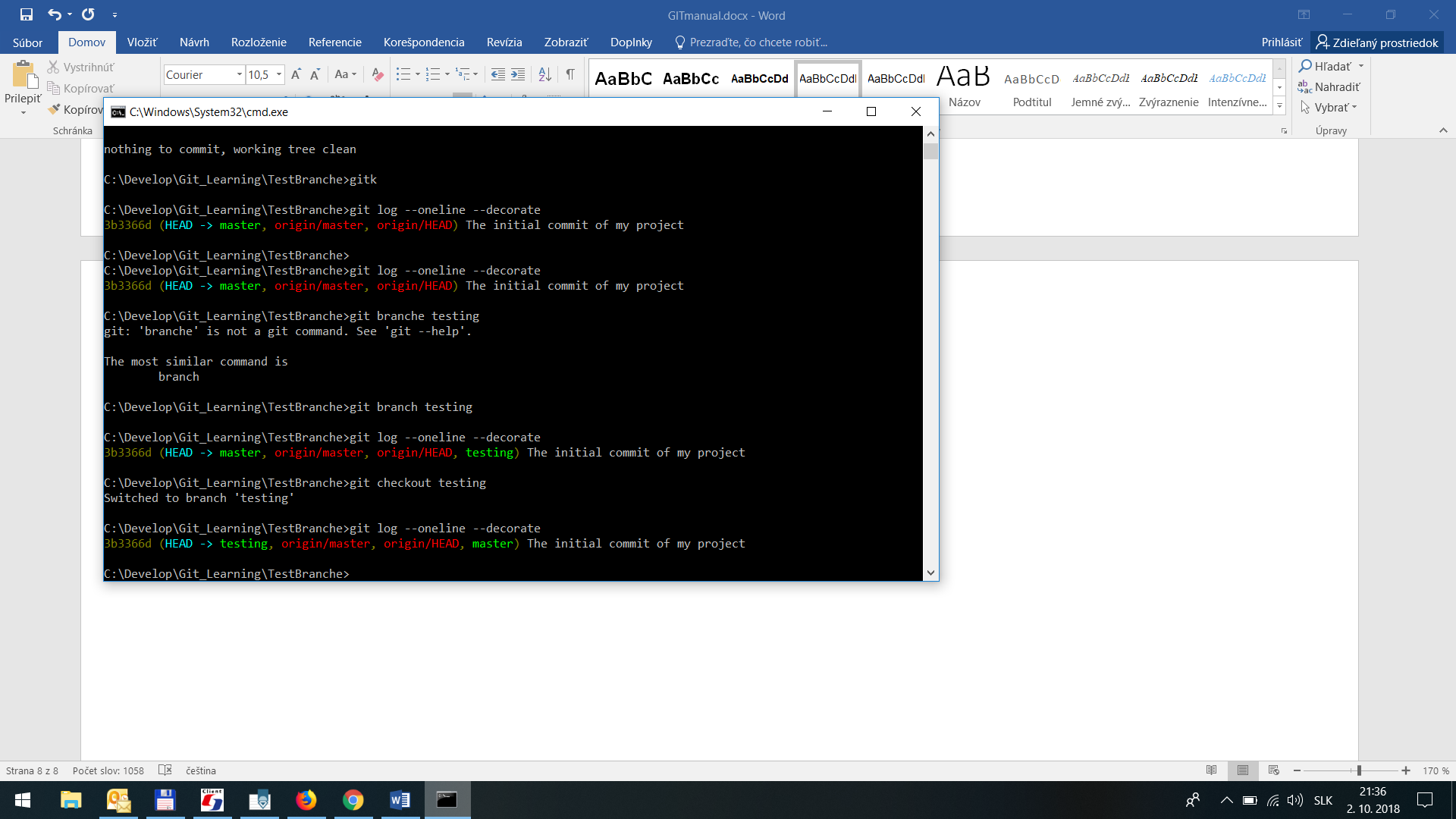
git log --graph --full-history --all --pretty=format:"%h%x09%d%x20%s"





Na inú branche sa prepnem pomocou:

git checkout testing





Urobím úpravu súboru a commitnem zmeny:

git commit -a -m "made a change"



Následne sa prepnem na branche master

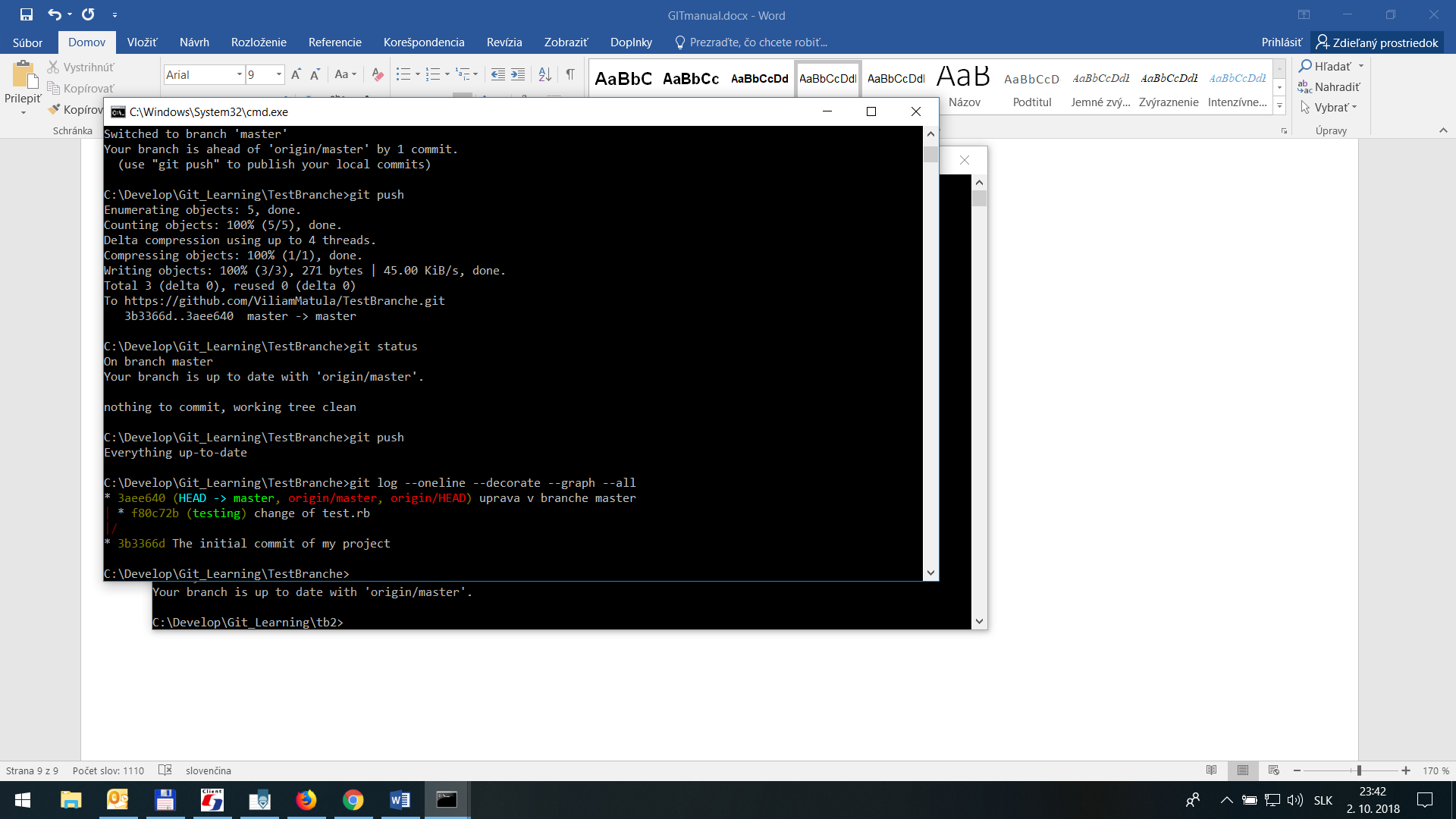
git checkout master

Urobím zmeny a zakomitujem

git commit -a -m "uprava v branche master"

Môžem si zobraziť aj jemne grafický výstup grafu:

git log --oneline --decorate --graph –all



Keď clonujem vzdialenú repository preniesla sa len aktálna branch. Aby som preniesol všetky dám

git clone <https://github.com/ViliamMatula/TestRemoteRepo.git> --all

Potom checkout-ujem potrebnú branch.

Môžem prípadne vyklonovať vybranú branch rovno:

git clone -b testing --single-branch <https://github.com/ViliamMatula/TestRemoteRepo.git> --depth 1

Alebo môžem vytvoriť branche a rovno sa na ňu prepnúť:

git checkout -b branchMojaNova

nahradzuje dva príkazy:

git branche branchMojaNova

git checkout branchMojaNova

Keď chcem zmergovať dve vetvy, prepnem sa na cieľ kam to cieli a zadám čo tam chcem dotiahnút:

git checkout master

git merge hotfix

Zmazanie branche:

git branch -d hotfix

Keď zmergujem dve brache dohromady, vytvorí sa merge commit

A