WORKSHOP

ARBETA LOKALT

Del 1 går ut på att bli lite familiär med att arbeta med git lokalt, utan att interagera med andra, och utan att kontakta eller skicka kod till en remote.

- Skapa lokalt repository
- Lägga till filer
- Commit
- Visa historik
- Visa diff
- Working copy, staging, repository
- Cleaning working copy
- Ignorera filer

Ber om ursäkt i förväg för Word som envisas med att ”rättstava” en del kommandon och saker man skriver. Jag har gått igenom övningarna ett par gånger, men det kan finnas fler fel. Copy-paste av kommandona till en prompt fungerar inte som det borde då vissa teckan är ersatta med typografiska motsvarigheter (ex. citattecken).

ÖVNINGAR

SKAPA NYTT REPOSITORY

Skapa en tom mapp, och skapa en README.txt enl...

echo "#mah-workshop-test" >> README.txt

git init

git add README.txt

git commit -m "first commit“

ALIAS

git log --graph --oneline

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

omständigt…. skapa alias

git config --global alias.lga "log --graph --oneline --all --decorate"

git lga

LÄGGA TILL FILER

echo "Hello world" >> README.txt

git status En fil modified

git commit (fungerar inte, måste lägga i stage)

git add .

git status

git commit -m "updaterad readme"

git status

Du ska nu vara i en ”clean” status.

COMMIT / VISA HISTORIK

git log

git diff xxxxxx..xxxxxx Räcker att man skriver första 6 tecknen av commit-hashen

git diff HEAD~1..HEAD

git diff HEAD~1..

LÄGGA TILL MER ÄN EN FIL

Skapa två nya filer, file1.txt och file2.txt

git add -u Funkar inte.. Filerna är nya inte uppdaterade befintliga.

git status

git add -A Lägger till ALLA filer

git status

git commit -m "lagt till nya coola saker"

UTVALD COMMIT

Ändra file1.txt

Ändra README.txt

git add file1.txt

git status

git commit -m "Fixat bug#1234“

git add README.txt

git commit -m "Fixat stavfel"

DELETE OCH RENAME

Radera file2.txt

git status

git add -u

git status

Döp om file1.txt till new_file.txt

git status Notera att det file1 står som deleted och new_file som untracked

git add -A

git status Notera att file1 numera står som renamed

git commit -m "Omorganiserat"

git status

git log

BACKA ÄNDRINGAR I WORKING DIRECTORY

Ändra i README.txt

git status

git checkout README.txt

git status readme.txt är nu återställd, och du har inga lokala ändringar

Töm readme.txt, och radera file1.txt

Skapa ny fil untracked.txt

git status

git reset --hard

git status workdirectory återställd till senaste commit – notera att untracked fortfarande är kvar.

git clean -f -d Force delete untracked files (se längre ner)

git log

BACKA COMMITS

git reset --soft HEAD~1

git status Notera att ändringarna från förra commiten har rullats tillbax, och ligger i stage

git commit -m "hej hopp"

git status

git reset --hard HEAD~1

git status Notera att ändringarna från förra commiten rullats tillbax helt och hållet

CLEANING

Skapa filer: temp1.txt, temp2.txt

git status

git clean

git clean -n ..Would remove…

git clean -f ..Removing…

git status

.GITIGNORE

mkdir logs

touch logs/log.txt

git status

Skapa filen .gitignore med innehåll:

/logs/*.txt

(Använd ”touch .gitignore” windows gillar inte att skapa filer som börjar med punkt) eller

echo /logs/*.txt >> .gitignore

git status

git add .gitignore

git commit -m "Added ignore-file"

TAGS

git tag v1.0

git tag

git log (taggen syns inte i loggen)

git log --oneline --decorate

git tag -a v1.0_withmessage

git tag

git tag -s v1.0_signed Signerade taggar kräver ev extra installationssteg och nycklar

git tag -v v1.0_signed

BRANCH / MERGE

Arbeta med lokala branches

Stashing

Merging branches

Rebasing commits

Cherry-picking commits

SKAPA NY BRANCH

git branch

git branch feature1

git checkout feature1

git lga Använder aliaset för (git log –oneline –decorate) ovan

echo "Feature1" >> README.txt

git commit -am "Lagt till feature1"

git lga

BRANCHA(!) AV FRÅN TIDIGARE COMMIT

git lga

git branch fix1 xxxxxxx (välj från tidigare commit)

git lga Notera att det skapats ny branch en bit bak

git checkout fix1

echo "fixar saker" >> README.txt

git commit -am "I will be deleted later"

git lga

DÖPA OM OCH RADERA BRANCH

git checkout master

(kör git lga mellan varje för att se ändringarna)

git branch -m fix1 bug1234 Notera att fix1 har döpts om till bug1234

git branch -d bug1234 GIT varnar att du kan tappa bort saker

git branch -D bug1234 branch bug1234 raderas

git lga

(checkout -b = git branch feature2; git checkout feature2)

git checkout -b feature2

echo "Feature2" >> README.txt

git commit -am "Lagt till feature2"

git lga

ÅTERSTÄLLA RADERADE COMMITS

git reflog (notera raderad commit HEAD@{3})

git branch bug1234 HEAD@{3}

git lga

git checkout bug1234

git show HEAD

type README.txt

STASHING (TEMPORÄRT SPARA ÄNDRINGAR)

git checkout feature2

echo "Ändra lite i Feature2" >> README.txt

git status En ändrad fil.

git stash

git status ändringen återställd

type README.txt Saknar föregående ändring

git stash list

git checkout bug1234

echo "En fix till i bug1234" >> README.txt

git commit -am "Fixa bug1234 på riktigt"

git checkout feature2

git stash apply notera att ändringen ovan är tillbaka

type README.txt

git stash list notera att ändringen fortfarande ligger i stash

git reset --hard HEAD

git status ändringen bortplockad igen

type README.txt

git stash pop

git stash list änderingen återigen applicerad, men denna gången borta ur stash

MER ÄN EN STASH

git stash

echo "Mer ändringar" > en_fil_till.txt

git status

git stash

--funkar inte.. Måste lägga till filen i stage.

git add en_fil_till.txt

git stash

git status (inga ändringar)

git stash list 2 saker stashade

git stash drop

git stash list senaste stashen raderad, en sak kvar

git stash branch feature2_additional Kortkommando för ”stash pop” till ny branch

git stash list

git commit -am "Lagt till ytterligare saker i feature2"

git lga

Its starting to get messy ☺

MERGING BRANCHES

git checkout master

git lga

git merge feature1 (fast forward)

git branch -d feature1 (safe to delete...)

git merge feature2_additional (conflict)

----titta på README.txt

git mergetool

--- beroende lite på vilket verktyg som är konfigurerat kan detta se lite olika ut, tanken är att readme totalt ska innehålla både feature1, feature2 och feature2-mod efter merge.

git status

git diff --cached

git commit -m "Merged feature2_additional into master"

---- städa bort ev. skräpfiler (xxxxx.orig)

REBASING CHANGES (NO CONFLICTS)

git lga

git branch feature3 v1.0

git checkout feature3

---- Editera file1.txt

git commit -am "Lagt till feature3"

git lga

git show feature3

git rebase master

git lga

git checkout master

git merge feature3

git lga

REBASING CHANGES (WITH CONFLICT)

git checkout bug1234

git rebase master

----kolla på README.txt

git mergetool

---- städa skräpfiler

git status (Rebase in progress)

git diff --cached

git rebase --continue

---- Lös resterande problem (git mergetool, samt städa)

git rebase --continue

git lga (se att ändringen är uppflyttad till master)

git checkout master

merge bug1234 (all branches merged, and can be deleted safely)

git branch -d feature3

git branch -d bug1234

git branch -d feature2

git branch -d feature2_additional

CHERRY-PICKING

git branch v1.0_fixes v1.0

git checkout v1.0_fixes

echo "Fix1" >> file1.txt

git commit -am "Lagt till fix1"

echo "Fix2" >> file2.txt

git commit -am "Lagt till fix2"

----- flytta enskild ändring till master

git checkout master

git cherry-pick xxxxxxx (välj fix1)

git lga (notera att fix1 förekommer två gånger)

--- kolla file1 och file2

git merge v1.0_fixes

git lga Wow.. we made a mess…