Git

version control

Git Verkefni FTK

Efnisyfirlit

Uppsetning og fyrstu skrefin

Skoðum tólin okkar

Til að lýsa *git* á sem allra einfaldastan máta er hægt að segja að það sé aðferð til þess að geyma ekki bara kóða, heldur líka breytingarnar sem gerðar eru á kóðanum yfir tíma. Kóðinn er geymdur í hólfi sem er kallað *repository* og er oft stytt sem *repo*. Byrjum þetta á einu léttu verkefni, en það er að búa til eitt repository. Þá munum við einnig kynnast skipanalínunni í fyrsta skipti.

Byrjum á því að keyra upp forrit sem heitir á Windows *Git Bash*, en *Terminal* á Mac og Linux. Ef Git er ekki til staðar er hægt að sækja það á git-scm.com. Athugið að ef notast er við CMD (command prompt) virka ekki allar skipanir sem hér eru teknar fram.

Þegar við erum komin inn í skipanalínuna er tilvalið að búa til möppu fyrir *repositoryið* okkar sem við höldum síðan áfram að nota í næstu verkefnum.

Þetta gerum við með `mkdir` skipununni:

```
$ mkdir verkefni
```

Að því loknu viljum við færa okkur inn í möppuna með `cd` skipuninni.

```
$ cd verkefni/
```

Þegar við erum komin inn í möppuna okkar getum við búið til fyrsta *repoið* okkar. Það er gert svona:

```
$ git init
```

Þá ættum við að fá til baka skilaboð sem líta svona út:

```
$ Initialized empty Git repository in ...
```

1.1 - Verkefni með git init

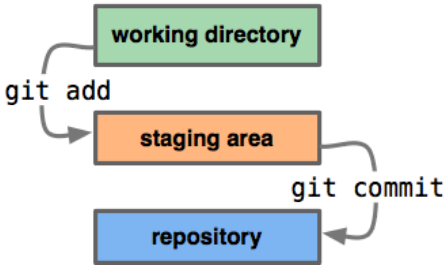
Núna höfum við farið yfir skrefin til þess að búa til *repo* með git. Þá er komið að fyrsta verkefni dagsins, sem er einmitt að nota `git init` til þess að búa til okkar eigið fyrsta *repo* sem við munum síðan halda áfram að nota það sem eftir er að verkefninu.

Rýnum dýpra

Þegar við komumst lengra í þessu munum við þurfa sífellt meira að skoða og fylgjast með breytingunum sem eru gerðar. Það er ein skipun fremri öðrum í þessum flokki og er sú skipun `git status.` Þessi skipun segir okkur hvað er að gerast einmitt núna í *repoinu* okkar. Ef þú hefur bætt við einhverjum skrám í *repoið* okkar, ættir þú að sjá lista af rauðum skráarheitum þegar Git status er keyrt.

```
$ git status
```

Núna ertu eflaust velta fyrir þér hvað þessi rauði litur þýðir. Rauði liturinn þýðir einfaldlega að git er ekki fylgjast með þessum skám. Afhverju er git ekki að fylgjast með þessum skrám? Það er vegna þess að við erum ekki búinn að biðja git um að gera það.



Næsta tól sem við munum læra að nota heitir `git add.` Þetta tól krefst frá okkur örlitla vitneskju. Við þurfum að vita muninn á *sviðunum* sem git býður upp á. Sviðin skiptast í þrjá flokka: *working directory*, *staging area* og *repository. Working directory* er mappan sem við erum að vinna í þessa stundina og kóðinn sem geymdur er á því sviði er ekki í öruggum höndum git. Við notum síðan `git add` skipunina til þess að færa kóðann okkar smátt og smátt yfir í *staging area* og þegar við erum komin með nægilega margar kóðabreytingar til þess að réttlæta *commit* notum við skipunina `git commit` til þess að færa tilbúna kóðan okkar í *repoið*.

Hér fyrir neðan má sjá hvernig á að nota `git add` til þess að færa skrár í *staging area*-að:

```
$ git add filename
```

Gott er að taka það fram að git add styður líka *glob* þannig hægt er að bæta við öllu *working directory-inu* og skrám í því með:

```
$ git add .
```

Staging area

Núna kunnum við að bæta við skrám í *staging areað* okkar. En hvað er *staging area*? Flott mynd var hér aðeins ofar en staging area er einskonar millisvæði þar sem hægt er að safna saman (með `git add`) öllum þeim skrám sem eru sendar með næsta *commiti*.

Lokaorð í bili

Það síðasta sem þú gerir við breytingarnar þínar er að ýta þeim í Git söguna fyrir *repoið* þitt. Það er gert svona:

```
$ git commit
```

`git commit` skipunin mun ein og sér gefa þér upp textavinnsluforrit (oft eitthvað á borð við vi) til þess að skrifa í einhver skilaboð um þetta commit. Það er mikilvægt að hafa þessi skilaboð lýsandi og ítarleg vegna þess að þau segja öðrum síðan til um hvað það var sem við breyttum með þessu committi.

Önnur gerð af commit skipuninni sem við getum keyrt er sjáanleg hér fyrir neðan:

```
$ git commit -m “Skilaboð”
```

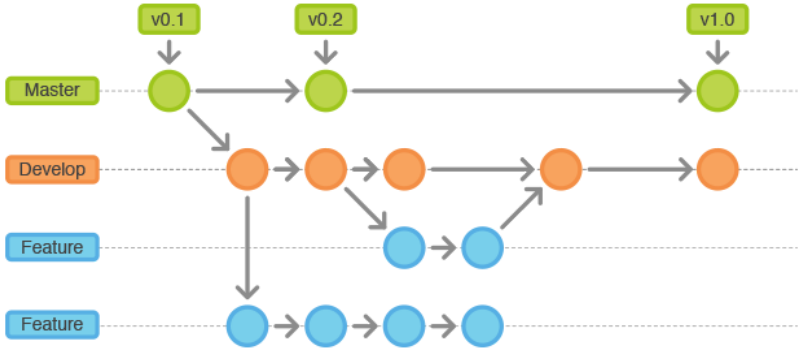
1.2 - Þegar allt er lagt undir

Nú þegar það fer að líða undir lok fyrsta kaflans ætlum við að vinna eitt stærra verkefni til þess að setja punktinn almennilega fyrir ofan i-ið. Við erum samt komin gífurlega stutt í námsefninu hingað til, sem takmarkar breiddina á því sem við getum gert. Við ætlum að búa til nokkur skjöl í *working directoryinu* okkar. Síðan ætlum við að færa þessar breytingar yfir í *staging area-ið* okkar með `git add` og að því loknu ætlum við að færa þessar breytingar yfir í *repo-ið* okkar með `git commit.` Gagnlegt er að endurtaka ferlið nokkrum sinnum til þess að við venjumst vinnuflæðinu áður en að við höldum lengra.

Grunnur í branching

Fáum smá yfirlit

Hvað er branching? Branching -eða *“að greina”* á góðri íslensku- er að brjóta skrif



kóðans niður í mismunandi *deildir*. Þetta er sniðugt til að mynda þegar það þarf að bæta nýjum eiginlegikum í forritið. Þá væri hægt að búa til sér branch fyrir þann eiginleika og allur kóðinn sem tengist honum væri þannig unnin í sér deild. Þetta gerir það að verkum að vinna getur áfram haldið í stofninum (kallaður master) án þess að vinna í auka eiginleikanum hafi áhrif á stofnin og stofninn á eiginleikan. Skipunin til þess að búa til nýtt branch er einfaldlega:

```
$ git branch branchname
```

Þetta býr til nýtt branch með nafnið *branchname*. Einnig er hægt að keyra:

```
$ git checkout -b branchname
```

Þessi skipun bæði býr til nýtt branch sem heitir *branchname* og færir þig yfir á það.

Greining

Þegar við búum til branch þá búum við til annan stað sem hægt er að vinna í verkefninu. Þetta er hægt að útskýra með því að ímynda sér að verkefnið sé tré en vinnan greinist út frá trénu eins og greinar þess. Skoðum hvernig er hægt að vinna með branches í grunnin og skellum okkur síðan í nokkur verkefni.

Að skoða allar greinar

Það er hægt að hafa margar greinar bara fyrir þig á þinni vél þó svo að þinn kóði sé einnig í einhverri kóðalaug. Þess vegna getur hver aðili fyrir sig verið með helling af brönchum og notfært sér eiginleika þeirra á sinni vél án þess að það hafi nokkurn tíma áhrif á þína vél. Til þess að skoða þær greinar sem þú ert með á þinni vél skrifar þú einfaldlega:

```
$ git branch
```

Til þess svo að komast á þessa grein og skoða eða vinna í verkefninu er hægt að keyra

```
$ git checkout branchname
```

Þarna geturu svo unnið í þínum viðbótum án þess að hafa áhrif á neitt annað branch. Sömu skipanir og við fórum yfir áðan gilda einnig þegar þú ert kominn inn í annað branch.

Country roads, take me home

Þegar þú hefur lokið við kóðabreytingarnar sem þú þurftir að framkvæma er greinin sem þú bjóst til búin að þjóna sínum tilgangi og það er kominn tími til að höggva hana

af, líkt og maður myndi höggva grein af deyjandi tréi. Fyrsta skrefið í því ferli er að færa okkur yfir í stofn greinina. Það er gert svona:

```
$ git checkout master
```

Þegar þú ert kominn aftur í *master* greinina getur þú skrifað:

```
$ git merge branchname
```

Þá blandaru saman kóðanum úr *branchname* í *master*.

Þetta er ekki bara ferlið sem þarf að fylgja til þess að sameina úr eitthverju *branchi* í *master* heldur er þetta ferlið sem maður fylgir til þess að sameina hvaða tvö brönch sem er saman. Það þarf bara að hafa í huga að branchið sem þú ert að sameina í er branchið sem þú ert núna staðsettur á og branchið sem þú ert að sameina úr fer síðan í *merge* skipunina.

Þegar vanta fer eldivið

Þegar greinin er síðan komin á endastöð er ekkert annað eftir en að losa okkur við hana fyrir fullt og allt. Það er einfaldlega gert svona:

```
$ git branch -d branchname
```

Þetta eyðir þessari grein ekki alveg, en við mælum með að vera viss þótt það séu en leiðir til að komast aftur í þessa grein. Alla vega í einhverja daga.

2.1 - Fáum okkur brönch

Búið til nýtt repo -alveg frá grunni, gott fólk. Búið síðan til einhverjar skrár og búið svo til nokkur brönch: feature-branch, stable-release, development og testingnewapproach. Síðan ætlið þið að breyta mismunandi hlutum í hverju branchi fyrir sig. Þetta er gott tækifæri til að negla það alminnilega í sig hvernig brönch virka og hvernig við skiptum á milli þeirra.

GitHub og aðrar kóðalaugar

Beint í djúpu laugina

Git fellur undir flokk forrita sem nefnast einu nafni “dreifð útgáfustjórn” vegna þess að í grunnin sér Git eingöngu um að halda utan um mismunandi útgáfur af kóðanum okkar. Það sem við ætlum að tala um í þessum kafla eru svokallaðar *kóðalaugar* sem halda utan um kóðann okkar í skýinu og bæta alls konar skemmtilegum viðbótum við Git. Kóðalaugin sem við munum fjalla lang mest um í dag heitir GitHub. GitHub er fremst

meðal Git kóðalauga og er óumdeilanlega vinsælust meðal forritara þessa stundina. GitHub fær þessar vinsældir sérstaklega vegna þess að þeir gæða Git alls konar eiginleikum sem það er ekki með til að byrja með.

Til þess að tengja þitt eigið repo við GitHub keyrum við þessa skipun:

```
$ git remote add origin git@github.com/username/repo.git
```

Núna er þetta repo tengt við Git þjóninn GitHub. Til þess að sækja eitthvað sem er nú þegar á GitHub þarf svo að keyra

```
$ git pull origin branchname
```

Og til þess að senda committaðar breytingar aftur á þjóninn þarf svo að keyra:

```
$ git push origin branchname
```

Ef þú vilt síðan setja ákveðinn þjón sem fyrirfram ákveðið val (hægt er að hafa marga þjóna bakvið eitt repo) þarf að keyra:

```
$ git push --set-upstream origin master
```

Núna er þessi grein með sjálfgefinn þjón sem kallast *origin*. Það þýðir að núna er hann tengdur við GitHub, vegna þess að við skráðum hann sem remote með nafnið *origin*.

3.1 - Okkar eigin litli garður

Núna er tilvalið að fara inn á GitHub og búa til aðgang þar ef þú átt slíkan ekki til nú þegar. Til þess að geta hýst eitthvað repo á GitHub þarftu að eiga aðgang þar. Verkefnið okkar er síðan að búa til Git repo á tölvunni okkar og færa það síðan yfir á GitHub. Að því loknu ætlum við síðan að búa til nokkrar greinar og framkvæma eitthverjar breytingar á greinunum okkar. Einnig er góð hugmynd að setja GitHub inn sem sjálfgefinn þjón fyrir *master* greinina okkar.

Það er mikilvægt að geta unnið með öðrum

Það kemur einstaka sinnum fyrir í forritun að við þurfum að vinna saman að einhverju sameiginlegu markmiði. Þá er maður ekki endilega fyrsti meðlimurinn sem gengur í verkefnið og hefur þar af leiðandi ekki þau forréttindi að fá að búa til repoið sjálft. Þá er líklega gagnlegt að kunna að sækja repo frá einhverjum öðrum. Það er einmitt eitthvað sem við ætlum að fara yfir núna. Þetta ferli kallast að *klóna repo* og er gert svona:

```
$ git clone https://github.com/gitnams/clone.git
```

Þegar nákvæmlega þessi skipun er keyrð er þetta verkefni klónað á tölvuna þína.

Fork your friends

Nú er kominn tími til þess að skoða annan eiginleika sem GitHub hefur bætt ofan á Git. Þessi eiginleiki heitir *fork* og er einstaklega gagnlegur þegar kemur að því að sækja og breyta kóða frá öðrum. Fork virkar á mjög sambærilegan máta við clone, en líkist samt sem áður branch á sama tíma.

Í sinni einföldustu mynd erum við að búa til nýtt repo á okkar GitHub account sem byggir á repo einhvers annars með því að klóna þeirra repo yfir í okkar. Við getum síðan breytt kóðanum og byggt ofan á hann eins og við viljum. Síðan býður GitHub meira að segja upp á einfalda leið fyrir okkur til þess að blanda *forkinu* síðan aftur í upprunalega repoið.

3.2 - Fork you very much

Vegna þess að *forkið* er ekki til í Git einu og sér getum við ekki notað það í gegnum skipanalínuna eins og við gerum með nánast öll önnur verkefni hérna. Þess í stað þurfum við að gera það í gegnum GitHub síðuna sjálfa. Verkefnið er sem sagt að fara inn á GitHub síðuna sem hýsir námsefnið okkar: gitnams og ýta þar á fork takkann. Þá mun afrit af repo-inu birtast á þínum aðgangi. Að því loknu ætlum við að framkvæma einhverjar breytingar þar á og commita þær í nýja repoið okkar.

Viltu plís toga frá mér

Annar mjög hentugur eiginleiki sem GitHub hefur kallast *Pull Request.* Þessi eiginleiki gerir manni kleift að bjóða hverjum sem er að taka við kóðabreytingum frá manni sjálfum og setja inn í sitt eigið upprunalega repo.

Tökum smá dæmi til þess að skilja þetta betur: Ég er að skrifa forrit og ég geymi það á GitHub. Þú sérð að ég er að vinna í þessu forriti og langar að bæta eitthverju kúl í það. Til þess að gera það klónar þú verkefnið og vinnur í kóðanum sem þér langar að bæta við og á meðan get ég haldið áfram að vinna í verkefninu óáreittur -jafnvel ekki vitandi að þú sért að vinna í verkefninu líka. Síðan þegar að kóðinn þinn er tilbúinn hendir þú einfaldlega í eitt *Pull Request* sem ég sé síðan á GitHub, hef tækifæri til að skoða og get síðan samþykkt eða hafnað eftir því hvort mér finnst það eiga við eða ekki.

Titill og lýsing

Github bíður uppá nánari lýsingu sem flott er að nýta sér. Til að nota þann eiginleika frá skelinni er commit einfaldlega keyrt með öðru -m fyrir lýsingu.

```
$ git commit -m “Titill” -m “Nánari skilaboð, sem er
betur séð að skrifa mikið magn af texta í”
```

3.3 - Kippum aðeins í hvort annað

Það sem við ætlum að vinda okkur í núna er smá vinnsla með pull requests. Til þess að skerpa á þessu ætlum við að nota repo-ið sem við forkuðum áðan og senda breytingarnar sem við framkvæmdum í verkefni 3.2 aftur í upprunalega repoið sem þið forkuðuð kóðann úr. Það er mjög gott að endurtaka þetta ferli nokkrum sinnum svo það gleymist ekki eins auðveldlega.

Við erum ófullkomnir (samt ekki)

Þrátt fyrir að við reynum að sjálfsögðu að gera okkar besta til að fara yfir eins mikið af efni og mögulega hægt er á sem skemmstum tíma eru nokkir hlutir sem við höfum einfaldlega ekki tíma til þess að fara yfir. Þessi kafli þjáist mest vegna þessa og ákváðum við þess vegna að minnast á þetta hér. Það eru nefnilega margfalt fleiri möguleikar sem GitHub býður upp á sem við náum ekki að fara yfir. Þar á meðal má nefna hópavinnu, samtök, gröf yfir vinnu á tíma ásamt allskonar félagsmöguleikum. Það er ekki af ástæðulausu að GitHub er oft kallað „Facebook fyrir forritara".

Fyrir lengra komna

Hvað er í gangi, hvert er ég kominn?

Núna er kominn tími til þess að stíga skrefinu lengra í skipanalínukunnáttu okkar og færa okkur askvaðandi yfir í Git vinnslu fyrir lengra komna.

Byrjum á að gleyma öllu sem við höfum lært í gegnum ævina um hvernig tími líður. Tími í Git er nefnilega ekkert endilega línulegt hugtak. Við byrjum ekki á A og endum síðan á Ö. Það er einmitt þarna sem „dreifingin" í „dreifð útgáfustjórn" kemur til leiks. Commit er nefnilega ekkert annað en bending á kóðasafn þitt eins og það var til á einhverjum ákveðnum tímapunkti. Núverandi staðsetning einhverrar greinar er því ekkert annað en bending á eitthvað commit. Síðan færist bendingin bara til eftir því sem við bætum fleiri og fleiri commitum í kóðagrunninn okkar.

Þetta er samt mjög frelsandi, vegna þess að þetta gerir okkur kleift að flakka til og frá í kóðagrunninum eftir okkar þörfum.

Fyrsta skipunin sem við lærum í þessum kafla heitir `git log` og gerir okkur kleift að sjá öll commitin sem hafa verið sett inn ásamt skilaboðunum sem voru skrifuð við hvert og eitt þeirra:

```
$ git log
```

Hérna er síðan dæmi um hvernig hvert commit lítur út:

```
commit 702c1f03c5ca89656b7eb7f8f93e650f04903951
Author: Ómar Högni Guðmarsson <omsigum6@gmail.com>
Date:   Tue Nov 29 20:20:49 2016 +0000

    now i did some real work
```

Til þess að skoða kóðann okkar eins og hann leit út í einhverju committi þurfum við að nota `git checkout` skipunina og skrifa síðan hluta af *commit hash*-inu sem við sjáum hér að ofan sem `702c1f03c5ca89656b7eb7f8f93e650f04903951`

Ef þið eruð að velta fyrir ykkur hvað commit hash er, þá er það hátturin sem Git geymir og flokkar öll commit. Hash á líka að vera einstakt. Líkurnar á öðru eru smávægilegar.

Í checkout þurfum við að setja inn annað hvort allt commit-hashið eða bara nóg af því til þess að það sé *unique*

```
$ git checkout 702c1f03c5
```

Með þessari skipun erum við að ,,kíkja á" hvernig okkar kóði leit út á þessum tíma. Ef keyrð er skipunin

```
$ git branch
```

rífum við ,,hausinn" af núverandi staðsetningu og setjum hann aftur niður þar sem commitið sem við vorum að skoða er til þess að undirbúa það fyrir nánari vinnslu.

```
* (HEAD detached at 702c1f0)
```

Þarna getum við skoðað greinina á þessum tímapunkti. Þetta getur verið rosalega nothæft ef einhverjum skrám hefur óvart verið eytt.

4.1 - Tímaflakk

Skoðið `git log`, einnig er hægt að keyra `git log --help` til þess að sjá hvers kyns upplýsingar skipunin getur gefið frá sér. Prófið síðan að keyra `git log` í öðru branchi til þess að sjá hvernig það hegðar sér þar.

En ég vil ekki allt

Núna erum við að fara að snerta á lausnum sem varða gífurlega sérstök tilfelli og er alls ekki líklegt að lenda í þegar þú vinnur í Git frá degi til dags. Skipunin sem við ætlum næst að fara yfir heitir `git cherry-pick`.

Cherry-pick er notuð undir mjög sérstökum kringumstæðum sem er erfitt að ímynda sér að kæmu upp, en koma merkilega oft upp þrátt fyrir það. Gefum okkur smá dæmi til þess að átta okkur á tilfellinu þar sem þetta ætti við:

Þú ert að vinna verkefni. Fyrst ert þú eingöngu að vinna á *master* branchinu þar sem þú bætir við kjarnaeiginleikum verkefnisins, þegar þeir eru komnir ansi langt ákveður þú að búa til nýtt branch þar sem þú vinnur að einhverjum nýjum eiginleika fyrir verkefnið þitt. Eftir nokkur commit í nýja branchið kemstu allt í einu að því að það er villa í fyrra branchinu. Þú lagar þá villuna með því að skipta snögglega yfir í *master* branchið og setja inn commit sem lagar villuna þangað. En ó, nei! Villan er ennþá til staðar í branchinu sem þú bjóst til fyrir nokkru síðan til þess að vinna í. Það er nákvæmlega þar sem `cherry-pick` kemur til leiks. Cherry-pick tekur við umtöluðum *commit höshum* og gerir manni kleift að færa þetta eina kommit sem lagaði vandamálið inn í aukagreinina sem þú vilt halda ótrauður áfram að vinna í.

```
$ git cherry-pick ##fix hash###
```

Núna ef þú skoðar git log inni í feature branch sérðu að þetta eina commit bætist við og búið er að laga villuna þína í feature-branch. Git cherry-pick er snilld til að sækja bara eitt commit sem þú þarft. Dæmið sem við settum upp er bara eitt notagildi fyrir þetta tól.

4.2 - Back to the future

Notaðu tólin sem við lærðum á hér fyrir ofan til þess að búa til repo, setja nokkrar skrár í það og eyða síðan einni þeirra í burtu. Haltu síðan áfram að vinna aðeins í skránum með nokkrum committum. Reyndu síðan að því loknu að bjarga skránni sem þú eyddir „óvart”.

Ég? Nei, þú!

Ómar og Eyþór vinna stundum verkefni saman, en það reynist þeim oft erfitt, aðallega vegna þess að Eyþór er stöðugt að gera stafsetningarvillur. Ómar og Eyþór rífast linnulaust yfir þessum villum sökum þess að Eyþór heldur því fram að Ómar hafi í raun verið ábyrgur fyrir stafsetningarvillunni. Sem betur fer fyrir Ómar er til hentugt tól til þess að rekja til baka villur í Git, en það tól heitir `git blame`. Git blame gerir okkur kleift að skoða hver gerði hvað í einstaka skrám. Þú keyrir einfaldlega:

```
$ git blame filename
```

Og þá kemur á móti

```
$ fb7e6909 (Eyþór Máni Steinarsson 2016-12-02 14:27:42
+0000 1) ég er mjg gópur as sleva
```

Þarna sést skýrt og greinilega hver ber ábyrgð á þessum stafsetningarvillum.

4.3 - lllt er öðrum ólán sitt að kenna

Við ætlum að tvíþætta þetta verkefni. Í fyrsta þættinum ætlum við að nota Git blame á eitthvað af repounum sem við erum búin að búa til á vélunum okkar. Í seinni þættinum ætlum við að reyna það sama, nema hvað við ætlum að gera það í umhverfi GitHub.

Ég er ekki að blokka, bara að hunsa þig

Git ignore er notað til þess að segja Git frá þeim skráartýpum sem þú vilt alls ekki að Git fylgist með. Þetta er aðallega notað fyrir stillingarskrár sem eiga bara við þann sem er að vinna á tölvunni sem forritið keyrir á eða lykilorð sem þú vilt ekki að dreifist um allt. Hérna getur þú séð dæmi um hvernig git ignore skrá getur litið út:

```
conf.py
```

Ef við viljum síðan bæta við fleiri hlutum í .gitignore þurfum við einfaldlega að bæta við skráarheitum í þennan lista. Hérna er dæmi um lengri lista:

```
conf.py
.env
IrrelevantFile.txt
passwordThatShouldNotGoOnTheInternet.txt
/node_modules/
```

Hérna neðst sjáum við síðan dæmi um hvernig möppur eiga að líta út í .gitignore skránni.

4.4 - .gitignore

Núna ætlum við að vinna eitt verkefni með .gitignore. Það sem við ætlum að gera er að búa til skrá sem gæti innihaldið einhverjar viðkvæmar notendaupplýsingar, t.d. tengiupplýsingar fyrir gagnagrunn. Við viljum að sjálfsögðu ekki fá neitt af þeim upplýsingum á GitHub og þess vegna setjum við hana í .gitignore. Hinsvegar viljum við að aðrir notendur sjái hvaða upplýsingar eiga að vera í skránni og í hvaða formi, þannig að við viljum hafa á GitHub einskonar template sem aðrir geta hermt eftir.

Árekstrar og ágreiningur

Við lendum í smá bobba

Vegna þess að allir gera einhvern tímann mistök eru til nokkur tól í Git sem hjálpa okkur að greiða úr þessum mistökum. Við munum fjalla um tvö helstu tólin í þessum kafla. Fyrst ætlum við að fjalla um `git revert`. Þetta tól er minna notað af þessum tveimur vegna þess að það hefur bein áhrif á Git söguna þína, sem er langt frá því að vera það sem maður vill.

```
$ git reset --hard commithash
```

Git eyðir þá öllum *commitum* í *branchinu* aftur að þessu committi. Sem er langt frá því að vera gott ef einhver annar en þú er að vinna í *repoinu*

```
$ git revert commithash
```

Í þessu dæmi býr Git til nýtt commit og færir hausinn þangað í staðin fyrir að eyða öllu.

Úps

Jafnvel þó það kæmi óvart fyrir að einhver hópmeðlimur keyrir óvart `git reset --hard` á *repoinu* er ekki öll von úti -þ.e. nema óvinur okkar allra, git garbage collector, sé búinn að vinna sig í gegnum repoið. Git geymir nefnilega líka fyrir okkur sögu um hvar við erum búin að vera í *repoinu* okkar. Þessi saga er síðan aðgengileg með `git reflog` skipununni.

```
$ git reflog
```

Þarna kemur fram listi yfir hvaða staðsetningu „hausinn" á Git er búinn að hafa. Þar ættir þú einnig að geta séð nýjasta committið og hvar þú varst þar á undan. Með þær upplýsingar í hendi getur þú búið til nýja grein út frá þeirri staðsetningu sem er síðan hægt að *merge-a* í *master*.

Það er þá gert svona:

Skrifað:

```
$ git reflog
```

Sem skilar meðal annars:

```
06d9dce HEAD@{14}: commit: Bætti við mikilvægri skrá
```

Síðan skrifað:

```
$ git checkout -b branch-save-files
```

Og að því loknu:

```
$ git merge 06d9dce
```

Hérna erum við samt aðeins farnir að hakkast með notkun á git. Þú vilt oftast ekki gera svona hluti en þegar kemur að því er betra að vita hverning á að gera það en ekki.

Það er stundum í lagi að hakka smá

Ef það er enginn búinn að setja inn commit á eftir þér og þú vilt breyta skilaboðunum sem þú settir inn með seinasta commiti er hægt að gera það svona:

```
$ git commit --amend -m “ný skilaboð”
```

Þá færast nýju skilaboðin yfir á gamla *committið*.

Merge conflict

Ef fleiri en ein manneskja eru að vinna í einni skrá og commita hana ekki í rökrænni röð miðað við hvora aðra kemur upp hið víðþekkta *merge conflict*. Þrátt fyrir að vera alræmd sem leiðinleg er tiltölega auðvelt að ganga frá *merge conflictum*. Allt sem þarf að gera er að keyra:

```
$ git status
```

Þá færðu lista af þeim skrám sem urðu fyrir árekstrinum. Síðan þarftu að fara aftur í skrána í hvaða textavinnsluforriti sem þú kýst að nota. Þá ætti skráin að líta nokkurn veginn svona út:

```
<<<<<<<<HEAD        sýnir þýna útgáfu af filenum
Kóði sem er á tölvunni þar sem þú ert



============
Sýnir kóða t.d. Frá github, eða annari grein



Commit hash >>>>>>>>>>>>>>>
```

Það eina sem þú þarft þá að gera er að taka til í þessum skrám með því að velja hvora breytinguna þú vilt halda í. Þá ætti kóðinn að líta einhvern veginn svona út:

Kóði sem er á tölvunni þar sem þú ert

Það eina sem þú þarft að gera að því loknu er að nota `git add` til að bæta skránum í *staging area* og síðan *commita* þeim með einhverjum fínum skilaboðum.

4.5 - Búum til merge conflict

Í þessu verkefni ætlum við að búa til *merge conflict* og laga það síðan. Það fyrsta sem við þurfum að gera er að fara á GitHub, búa til nýtt *repo*, og setja í það tómt README.md skjal. Síðan ætlum við að sækja *repoið* á vélina okkar með Git clone. Að því loknu er gott að gera einhverjar breytingar á skjalinu og *commita* síðan þær breytingar aftur inn. Næst er að fara á GitHub á netinu og breyta skjalinu þar, án þess að *push-a* eða *pull-a*. Þegar hingað er komið förum við aftur á vélina okkar og skrifum `git push`. Þá lendum við í týpísku *merge conflict-i* og verkefnið er núna að laga úr því.

Smíðum sundlaug

Uppsetning Git þjóns

Git samtökin sjá sjálf um að búa til og viðhalda leiðbeiningum sem varða uppsetningu Git þjóna. Þær eru aðgengilegar á heimasíðu Git: git-scm.com. Þeir fara mun ítarlegar en við í uppsetningu Git þjóns, en þeir ítarþættir varða aðallega hert öryggi, sem er ekki eitthvað sem við erum að velta okkur mikið upp úr í dag.

Það eina sem þarf að vera til staðar til að leysa úr læðingi Git skrímslið sem býr í brjósti allra Linux véla er uppsett SSH tenging með SSH lyklum í stað, eða samhliða, tengingu sem reiðir enn á löngu úreltu lykilorði fornaldar.

Fyrsta skrefið er síðan að sjálfsögðu að tengjast vélinni:

```
$ ssh username@example.example
```

Þegar við erum komnir með tengingu við vélina okkar þurfum við bara að búa til möppu:

```
$ mkdir verkefni.git
```

Síðan færum við okkur inn í möppuna:

```
$ cd verkefni.git
```

Og búum síðan til bert Git *repo* með skipuninni:

```
$ git init --bare
```

Að því loknu þurfum við bara að fara aftur í heimavélina okkar og tengjast þjóninum svona:

```
$ git remote add newserver
username@example.example.com:/home/verkefni.git
```

Núna er tengingin komin á og við getum sett Git kóðann okkar á þjóninn.

Einföld og tilbúin lausn.

Önnur mjög fullkomin lausn til að eiga kóðann þinn sjálfur heitir GitLab en það gefur líka vefviðmót og mjög einfalda og góða leið til þess að búa til og fylgjast með notendum. GitLab er margfalt notendavænni upplifun og býður upp á miklu breiðari notkunarmöguleika en gamli góði Git þjónninn.

Annað efni

Git handbókin

Þetta er opinbera Git handbókin, skrifuð af þeim sem viðhalda Git skipanalínunni og er ætluð fyrir þá sem kunna nú þegar kunna á Git sem hækja þegar maður man ekki eitthvað eða vantar að fletta upp til þess að greina úr einhverjum vandamálum.

GitHub Hjálp

GitHub hjálp er miklu aðgengilegri en Git handbókin sem aðstoðartól fyrir byrjendur. Við mælum eindregið með því að þið kíkið þangað til þess að skerpa á Git og GitHub kunnáttu ykkar eftir þessa vinnustofu. Einnig hefur þessi hjálp eitthvað sem Git handbókin hefur ekki og það er leiðbeiningar fyrir sérstaka eiginleika GitHub

Hjálparflaggið --help

Git býður upp á flagg sem kallast „`--help`”. Þetta flagg getur farið á eftir hvaða skipun sem er og þá sýnir Git hjálparupplýsingar fyrir skipunina sem kemur á undan flagginu.