اطلاعات بیشتر در باره گیت هاب
{https://sabzlearn.ir/blog/what-is-github-2/}

البته این قسمت بر گرفته از ویکی پیدا فارسی است. و فعلا دوست داشتم که قرار بدهم.
دستورها
برای راهنمایی گرفتن دربارهٔ یک دستور:

```
git help
```

برای اجرای تنظیم‌ها از config استفاده می‌شود. جزئیات استفاده از این دستور را می‌توان با دستور زیر به‌دست آورد.

```
git help config
```

دستور ایجاد پوشه اصلی:

```
git init <آدرس پوشه اصلی>
```

دستور استفاده از یک گیت از قبل ساخته شده:

```
git clone ssh://<user> @ <host> /path/to/repo.git
```

استفاده می‌شود. دستور اضافه کردن فایل جدید:

```
git add <نام فایل>
```

دستور حذف فایل

```
git rm --cached <نام فایل>
```

دستور مشاهده وضعیت:

```
git status
```

بررسی تفاوت‌ها در کد نوشته شده و اعمال نشده:

```
git diff
```

این دستور تغییرهای اعمال شده و ثبت نشده را نشان می‌دهد. دستور اطلاع از تاریخچه

```
git log
```

دستور قراردادن تغییرها:

```
git push
```

دستور اعمال تغییرها:

```
git commit
```

دستور دریافت تغییرها از روی تاریخچه:

```
git pull
```

دستور الحاق برچسب:

```
git tag
```

دستور بازگرداندن یک فایل به آخرین وضعیت ذخیره شده:

```
git checkout HEAD <نام فایل>
```

برای برگرداندن تغییرهای یک اِعمال خاص:

git revert <commit-id>

اصلاح یک اِعمال:

git commit -amend

سلام در اینجا من تصمیم دارم هر آنچیزی که خودم از GitHub را قرار یاد بگیرم و اینکه برای خودم نیاز میدانم که باید بنویسم را می نویسم و فعلا هم از روی ویدیوی های درس جادی جلود میرم و التبه از یک تعداد منابع دیگر :) البته قرار است که اینها را در گیت قرار می دهم که بعدا برای خودم و دیگرانی که نیاز دارن قابل استفاده باشد.

درس اول بر گرفته از درس جادی در فرادرس
گیت (git) را لینوس توروالدز (Linus Benedict Torvalds) به سوئدی نوشته: گیت ابتدا برای توسعهٔ لینوکس به وجود آمد و اکنون پروژه‌های فراوانی از آن الهام گرفته‌اند. هر دایرکتوری کاری در گیت یک مخزن کامل با تاریخچهٔ کامل تغییرها و قابلیت بازنگری آن‌ها است و برای کار با آن نیازی به دسترسی به شبکه یا سرور مرکزی وجود ندارد. گیت لزما برای کسی نیست که برنامه نویس هستن برای کسی که پایانامه، کتاب و ... کارهای دیگر هم میخواهند انجام دهند خیلی کارایی دارد و مفید است.
گیت (git) یک (distributed Verison Control System) است یعنی (VCS) خوب حالا بخواهیم فارسی توضیح بدهیم یعنی چی وقتی من شما پایانامه، مقاله یا سایت یا هر کاری مشابه دیگر میخواهیم طراحی کنیم به چی مشکلی می خوریم
پروژه را شروع می کنیم یک عالمه فایل دارد آنها را قرار می دهیم در یک Directory بعدش یک مقداری روی پرژه کار می کنیم یک چیزه دیگه درست می کنیم به اسم New Folder جدیده را کاپی می کنیم داخل آن بعد که کار ما تمام شد یک چیزی دیگری درست می کنم به اسم final پروژه را کاپی می کنیم داخل آن بعد از یک مدت final هم یه تغیرات میخواد می نویسم new بعد به همین ترتیت احتمالا پشت سر هم به تغییرات نیاز دارم که یک زمانی نمی فهمیم که چی کار می کنیم بعدترین حالت اینه که بفهمیم این یکی اشتباه داشت باید برگردیم به اون قبلی و در پوشه بندی که ما انجام دادیم نمی دانیم که قبلی آن کدام بوده و یا اینکه چند نفر باشیم روی یک پروژه کار کنیم یکی میخواهد روی یک فایل کار کند و یکی دیگر هم میخواهد روی یک فایل دیگر disterbuted ها درست شدن که چند نفر بتونن باهاش کار کنند یک ایده واقعا فوق العاده است .که بهترین شون گیت (git) است.
گیت پروژه ها را قسمت قسمت می کنه و میگه میخوای پروژه انجام بده بیا انجام بده و میتوانی از ورژن فعلی به قبلی دسترسی داشته باشی و همینطوری میتوانی ادامه بدهی. خوب یک مقدار توضیحات را گفیتم حال بریم سمت نصب گیت(:

درس دوم.

#git init

وقتی ما در یک repositeroy هستم و میخواهم که شروع کار با گیت را انجام بدهیم باید دستور اول مون git initباشد یعنی که شروع تنظمات اولیه را انجام دادن و این directory تحتی کنترول گیت است. اگر ما بخواهیم که directory خود را نگاه کنیم از دستور

#ls -ltrha

میتوانیم ببنیم که آیا directory به اسم گیت درست شده یا خیر. در لینوکس و مک اگر که اول یک فایل (.) باشد اون فایل را نمایش نمی دهد برای همین این دستور ما می توانیم که فایل خود را که تقریبا میتوانیم بگویم پنهان شده را ببنیم.

#ls -ltrha .git

میتوانیم که داخل فایل گیت خود را ببنیم.

#git status

برای نشان دادن وضعیت بکار می رود .

گیت با یک مفهوم کار می کند به اسم commit و stage در واقع میگه شما یک directory دارینند که فایل ها اینجا هستن، فایل ها میتوانند تغییر کنند وقتی فایل ها تغییر کنند شما آنها را می برید به یه وضعیتی به اسم stageیعنی آماده ای یه چیزی به اسم commite کردن .

my file ---> stage ---> commite

فایل را میگم تغییر کرده یا جدید هست اش به یه دستور می بریمش به مود stage یعنی اگر آنجا status بزنیم گیت می فهمه که این stage است و با یک دستور دیگه commite اش می کنیم فایل که commite میشه در واقع میره در آرشیو (git.) و یادش نگه میداره که این فایل state اش اینه و اینگار تغییر نکرده.

#git add (file name)

#git add test.txt

برای اضافه کردن فایل. البته نباید از قوس باز یا بسته استفاده کنیم از قوس ها برای فهم بیشتر است. برای اینکه بعدا اشتباه نکنیم نمونه دوم نمونه اصلی است.

#git add "file*"

این برای این است که شروع فایل که مثلا با file هر چیزی شروع شدن را اضافه کن

#git add "*.java"

برای مثلا تغییرات همه فایل های که با java ختم می شوند یا هر چیزی دیگری

#git add -A

برای تغییرات همه فایل های که است را قبول کن. باید حرف A باید بزرگ باشد.

#git status

برای اینکه بفهمیم که در چی حالت یعنی وضعتی قرار دارمی.

#git commit -m "create a ..."

انیجا تغییرات که آوردیم را می نویسم اگر یه زمانی کسی log این ار نگاه کرد بفهمه که چی تغییرات ایجاد کردیم.

درس سوم

#git log

این دستور log کار های ما را نشان میدهد که چی کارهای انجام دادیم. مثلا میگه چندتا کامیت داشتیم هر کامیت هم یک کد یونیک پیدا می کند.

فایل ها ما اول Untracked هستن که اصلا گیت مواظب آنها نیست بعد add می کنیم tracked می شوند، تغییر پیدا کردن یا جدید هستن. بعد هم میتوانیم کامیت شان کنیم که می روند در محیط .stage

نکته: تقریبا هیچ کس نمی توانه اداع کند که تمام گیت را بلده چون خیلی بزرگه هر قسمت هم که گیر کردیم یا به مشکل خوردم در انترنیت جستجو کنیم.

درس چهارم

#git diff

نشون دهنده اینه که یه چیزی با یه چیزی دیگری وضعیت اش چی است.

#git diff HEAD

میگه که الان گیت کجا است. تغییرات فایل ها برای ما نشان میدهد و با ما نشان میدهد که یه تغییرات در فایل ها انجام شده است. دقیقا میگه که کدام قسمت ها تغییر کرده مثلا با a---) قدیمی (نشون میده که قبلا این را دشتید و با b+++) فعلی (فعلا این را دارید.

#git diff --staged

نشون بده که آنهای که در stage هستن چیشون عوض شده نه اینکه همه فایل ها را البته میتوانیم یک فایل خاص را هم بگیم.

#git reset (file)

یه چیزی را میتوانیم از stage بیاریم بیرون مثلا از از تغییرات که در فایل انجام دادیم راضی نیستیم و خوشم نیامده. اگر من فایل خود را commite کنم جز این فایل همه را فایل ها را می بره به حالت stage و حال اگر پروژه را commite کینم جز این این فایل آنها را که خواستیم commite می شوند.

#git checkout -- (file name)

اگر کلا فایل را خراب کردم و میخواستیم بر گردیم به قبلی از این دستور استفاده می کنیم. فایل مورد نظر را بر می گردونه به اون commite آخر که انجام دادیم. بی خیال همه تغییرات میشه که داده باشیم.

#git commite

یه ادیتور خالی میاد و میگه commite چی میخوای باشه در چی که دوست داشتی میتوانی بنویسی.

درس پنچم.

آدرس سایت برای توضیحات اضافی در باره branch

https://sokanacademy.com/academy/courses/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%DA%AF%DB%8C%D8%AA/%D9%81%D8%B5%D9%84-%DB%B1-121/%D8%A2%D8%B4%D9%86%D8%A7%DB%8C%DB%8C-%D8%A8%D8%A7-%DA%A9%D8%A7%D8%B1%D8%A8%D8%B1%D8%AF-branch-%D8%AF%D8%B1-git

زمانی که با استفاده از گیت یک ریپازیتوری جدید می‌سازیم، به صورت خودکار یک برنچ (شاخه) تحت عنوان masterساخته می‌شود که نقش شاخهٔ اصلی ریپازیتوری مذکور را بازی خواهد کرد و هر کامیتی که انجام دهیم نیز روی این شاخه اِعمال خواهد شد و این در حالی است که معمولاً تیم‌های نرم‌افزاری از این شاخه به عنوان نسخه‌ای از نرم‌افزار استفاده می‌کنند که قرار است روی سرورهای اصلی دیپلوی گردد. با این تفاسیر، منطقی به نظر می‌رسد که این بِرَنچ به عنوان فضای آزمون و خطا در حین کدنویسی قملداد نشده بلکه فضاها یا بهتر بگوییم شاخه‌های فرعی دیگری ساخته و در آن‌ها اقدام به توسعهٔ فیچرهای جدید نموده سپس آن‌ها را با شاخهٔ مَستر ادغام نمود.

#git branch

برای ما نشان میده که روی چی برنچی هستیم و یا اینکه چی برنج های داریم.

#git branch (branch name)
#git brach test

یک برنچ جدید درست می کنیم و روی آن کار می کنیم تا بعدا با پروژه اصلی merge کنیم.

#git checkout (branch name)
#git cheackout test

با این کار ما روی برنچ دیگری می توانیم بریم و روی آن کار خود را شروع کنیم.

#git merge test

با این کار تغییرات برنچ قبلی را به ماستر merge می کنیم. التبه باید روی برنچ ماستر باشیم و بعد این کار را انجام بدهیم.

درس ششم.

#git rm (file name)
#git rm test.html

برای حذف فایل هم گیت و هم از فایل سیستم.

#git branch -d (file name)
#git branch -d test

اگر اینکه بخواهیم برنچ که ساختیم را حذف کنیم و البته دیگه نیازی بهش نداشته باشیم از کامند بالا استفاده می کنیم.

درس هفتم.

#git clone (https"//(address)

برای clone کردن کد استفاده می کنیم.

Origin

یه اسمیه که وقتی ما یک چیزی را از یه سایت clone می کنیم معمولا اسم اش را میزارن origin ما هم که masterهستیم.

#git push origin master

برای اینکه تغییرات خود را بتوانیم پوش کنیم میگویم که پوش کن روی origin برنچ master ما را یعنی اینکه تغییرات اعمال شده ما را.

#git push -u origin master

همون کار قبلی اما این بار دیگه نیاز نیسی کهoriginرا تایپ کنیم دیگه در حافظه خودش ذخیره می کند ، master

#git pull origin master

برای اینکه تغییرات انترنیتی خود را روی پرژه شخصی اعمال کنیم pull می کنیم. یعنی اینکه هر چی تغییرات که در originاست را بردار بیار روی ماستر.

درس هشتم.

#git remote add origin (address https://)

با remote ها کار می کنیم.

#git remote add origin https://....

میگه من یک origin به این آدرس اضافه کردم.

#git remote -v

پر حرفش می کند یعنی اینکه به کجا pull یا اینکه push می کند را نمایش می دهد.

بعضی موقع ما یک فایل را همزمان تغییر بدهیم به یک سری مشکلات می خوریم به طور مثال یک فایل من در کلوید تغییر دادم و یکی دیگه در فایل شخصی خودش زمانی که بخواهد push کند به مشکل می خورد بهتر است pullکنیم بعد تغییرات را نگاه کنیم اگر از اون بهتر بود یا از ما بهتر بود تغییرات را انجام داده و بعد دوباره push کنیم.

درس نهم.

#git show (uniqe log code)

برای نمایش تغییرات log را ببنیم. اینکه در این قسمت از commite ما چی تغییرات ایجاد شده است.

#git tag

مثلا اینکه ما ویرژن گذاری کرده باشیم نمایش می دهد که در کدام ویرژن هستیم.

#git tag -a v2.0 -m "this is a first version"

اینکه ما مشخص می کنیم این دقیقا کدام ورژن از برنامه ما است.

#git tag -a v1.8 (uniqe log code)

با دییدن log ها و گرفتن uniqe code آن هم میتوانیم تعیین کنیم که این قسمت شامل کدام ویرژن است.

#git tag -l "v*"

هر چی tag کی اولش v است را برای من نمایش بده.

#git show v1.8

نشان میده log های این قسمت از ویرژن ما را.

#git push origin v1.8

مثلا اگر من git push را میزدم هیچی از tag ها پوش نمی کرد اما با این کار ما تغییرات tag را هم پوش می کنیم.

#git push origin --tags

این روش مرسوم تر از روش قبلی است. اما ورژن گذاری مثل برنچ نیست که بتوانیم مثلا در ورژن v1.8 تغییرات ایجاد کردیم را میرج کنیم برای این کار می توانیم دوباره برنچ بسازیم و تغییرات v1.8 را با هم میرج کنیم.

درس دهم.

#gpg --gen-key

برای ساختن هش های که در log برای تغییرات استفاده می شوند استفاده می شود که بعدا هیچ کسی تغییرات یکی دیگه را فیک نکند.

#gpg --list-keys

لیست از key ها را برای ما نمایش می دهد.

#git config --global user.name

برای نمایش اینکه اسم شما چی است.

#git config --global user.signingkey

این دستور برای نمایش signingkey بکار می رود.

#gpg --list-secret-keys

برای اینکه همه چیزهای مخف خود من را نمایش بده.

#gpg --list-secret-keys --keyid-format LONG

شبیه دستور بالا است اما برای اینکه طولانی تر باشد.

#git tag -s v2.1 -m "this a test"

با s- که اضافه می کنیم می گوییم که tag بزن و sigin اش با secret gpg و این کار باعث امنیت بیشتری می شود.

#git tag -v v2.1

نمایش log فقد با کد gpg خود شما نمایش داده می شود.

درس یازدهم.

{https://sokanacademy.com/academy/courses/%D8%A2%D9%85%D9%88%D8%B2%D8%B4-%DA%A9%D8%A7%D8%B1%D8%A8%D8%B1%D8%AF%DB%8C-%DA%AF%DB%8C%D8%AA-%D8%A8%D8%B1%D8%A7%DB%8C-%D8%A8%D8%B1%D9%86%D8%A7%D9%85%D9%87-%D9%86%D9%88%DB%8C%D8%B3%D8%A7%D9%86/%D8%B4%D8%B1%D9%88%D8%B9-%DA%A9%D8%A7%D8%B1-%D8%A8%D8%A7-%DA%AF%DB%8C%D8%AA/git-blame}

اطلاعات بیشتر در باره blame .

دستور git blame یک ابزار عیب یابی همه کاره است که کاربرد های زیادی دارد. عملکرد سطح بالای git blame، نمایش meta data ی نویسنده ی هر خط از کد های Commit شده در یک فایل است. این دستور برای بررسی یک نقطه مشخص در تاریخچه ی یک فایل و همچنین به دست آوردن آخرین نویسنده ای که خطی را اصلاح کرده است، استفاده می شود.

#git help blame

نمایش می دهد که blame چی کار می کند.

#git blame fileName

برای نمایش همه فایل.

#git blame fileName -L8 (L write which line number)

برای نمایش دقیق چی خطی.

#git blame fileName -L8,13

اطلاعت خط 8 تا 13 را نیاز دارم.

#git bisect start

دستور git bisect در حقیقت مخفف git binary search commit هست. ما وقتی ازین دستور استفاده می‌کنیم که در روند توسعه پروژه به باگ خوردیم و حالا می‌خوایم بدونیم که این باگ چه زمانی در کدوم کامیت پیدا شده و از طرفی کدوم ورژن ازrepository مون bug free هست git bisect .اینکارو با بررسی کامیت‌ها برامون انجام میده. برای معلومات بیشتر به لینک زیر مراجع کنید.

{https://virgool.io/@shadishirinbeik/%D8%A8%D8%A7-git-bisect-%D8%AA%D9%88%DB%8C-%DB%8C%D9%87-%DA%86%D8%B4%D9%85-%D8%A8%D9%87%D9%85-%D8%B2%D8%AF%D9%86-%D8%A8%D8%A7%DA%AF-%D8%B1%D9%88-%D9%BE%DB%8C%D8%AF%D8%A7-%DA%A9%D9%86%DB%8C%D8%AF-agcs3nkn3xqk}

#git bisect bad

میگم این قسمت بده

#git bisect good (log hash)
#git bisect good 1f13490b3ae89e983bch

با log مشخص می کنیم که این قسمت کد خوبه است.

#git bisect good

با دستور میگم که این قسمت خوبه بعد به همین ترتیب پشنهاد برای ما میدهد که آیا قسمت خوبه یا نه با این دستور تاید می کنیم که این قسمت خوبه.

درس دوازدهم.
اطلاعات در باره gitLab
فورک(Fork)
معنی: کپی از یک مخزن برای آزمایش تغییرات.
توضیح: استفاده معمولاً در پروژه‌های متن‌باز برای آزمایش تغییرات بدون تأثیر بر پروژه اصلی.
اگر که بخواهیم یک شخصی پروژه تعریف کرده و ما هم روی آن پروژه کار کنیم البته بدون دسترسی آن پروژه را forkمی کنیم