معرفی انواع نرم‌افزار

۱ -نرم‌افزار سیستمی(System Software)

نرم‌افزارهای سیستمی، نرم‌افزارهایی هستند که مدیریت و کنترل کلی سیستم‌های کامپیوتری را انجام می‌دهند. این نوع نرم‌افزارها به‌طور مستقیم با سخت‌افزار و منابع سیستم تعامل دارند و محیط مناسب برای اجرای نرم‌افزارهای کاربردی (مانند برنامه‌های کاربردی، بازی‌ها و غیره) را فراهم می کنند. سیستم عامل و درایور ها نمونه ای از نرم افزارهای سیستمی هستند.

۲-نرم‌افزار اپلیکیشن

رایج ترین نرم افزاری که اکثر ما از آن استفاده می کنیم، اپلیکیشن ها هستند. اپلیکیشن ها که نوعی از نرم‌افزارهای کامپیوتری هستند، طیف گسترده ای دارند. این اپلیکیشن‌ها معمولاً برای تلفن‌های همراه، تبلت‌ها، کامپیوتر های شخصی و سایر دستگاه‌های الکترونیکی توسعه داده می‌شوند. اپلیکیشن‌های موبایل، وب اپلیکیشن ها و بازی های کامپیوتری نمونه ای از این نوع نرم افزار ها هستند.

۳ -نرم‌افزار برنامه‌نویسی(Programming Software)

شاید شما هم با زبان هایC ، Javaو Python که برای توسعه برنامه ها، ساخت برنامه های نرم افزاری و تجزیه و تحلیل داده ها استفاده می شود، آشنا باشید. کد نویسی این زبان ها، توسط نرم افزارهای برنامه نویسی انجام می شود. نرم افزار برنامه نویسی از چند بخش ساخته شده اند:

- اسمبلرها:

هنگامی که دستورالعمل ها را در برنامه خود تایپ می کنید، اسمبلر آنها را به شکلی تبدیل می کند که برای پردازنده کامپیوتر شما قابل درک باشد.

- کامپایلرها:

کامپایلرها که شناخته شده ترین بخش در میان اجزای نرم افزار های برنامه نویسی هستند، کدهای نوشته شده را تجزیه و تحلیل کرده و به زبان سطح پایین مانند زبان اسمبلی یا زبان ماشین تبدیل می کنند.

- Debuggers:

دیباگر ها، برنامه ها را برای خطاها و اشکالات در برابر مجموعه ای از استانداردهای تعریف شده که یک نرم افزار برنامه نویسی مشخص از آن پیروی می کند، آزمایش می کند.

مفسران :(Interpreters)مفسر یک کد سطح بالا را به نسخه ای که می تواند توسط ماشین پردازش شود، تفسیر می کنند

۴ -نرم‌افزارهای تحلیلی:(Analytical Software)

این نرم‌افزارها برای تحلیل و پردازش داده‌ها، مدیریت اطلاعات و انجام محاسبات پیچیده مورد استفاده قرار می‌گیرند. نرم‌افزارهای مدیریت پایگاه داده و نرم‌افزارهای تجزیه و تحلیل داده نمونه ای از این نوع نرم افزار ها هستند.

همچنین، نرم‌افزارها به دسته‌بندی‌های دیگری نیز تقسیم می‌شوند که هر کدام کاربرد های خاص خودش را دارد. این دسته‌بندی‌ها می‌توانند بر اساس نوع مالکیت (نرم‌افزارهای متن باز و بسته)، نوع استفاده (تجاری و غیرتجاری)، میزان پیچیدگی و غیره باشد.

توسعه نرم‌افزار چیست؟

تمامی مراحلی که برای تبدیل ایده به نرم‌افزار طی می‌شود ساخت نمونه اولیه می‌گویند.

پس از ساخت نمونه اولیه(MVP) تمامی مراحلی که شامل افزودن فیچر جدید و رفع باگ می شود را توسعه نرم‌افزاری می‌گویند.

توسعه نرم‌افزاری عملی بدون پایان می‌باشد و دائما در حال تکرار می‌باشد.

توسعه نرم‌افزاری در واقع چه می‌کند؟

چرخه کلی توسعه نرم افزار

منظور از توسعه نرم افزار چیست؟

مراحل توسعه نرم افزار در شرکت های مختلف شامل چند مرحله اساسی است. این مراحل شامل برنامه ریزی، جمع آوری نیازمندی ها، طراحی، تست و تضمین کیفیت، توسعه، انتشار و نگهداری می باشد. هر مرحله دنبال کننده یک هدف نهایی است و هر یک به عنوان پله ای برای اقدامات بعدی عمل می کند.

مدیریت توسعه نرم افزار:

استاندارد بودن مراحل توسعه نرم افزار با توجه به موارد زیر در مدیریت صحیح پروژه موثر است:

۱-چارچوبی برای ردیابی و اندازه گیری پیشرفت در پروژه های توسعه نرم افزار مشخص می کند.

۲-صرف نظر از بزرگی یا کوچکی پروژه، کنترل مدیریت پروژه را تسهیل می‌کند و به پیشرفت توسعه نرم‌افزار کمک می‌نماید.

۳-فعالیت‌های تیم را هماهنگ می‌کند؛ در نتیجه همه اعضای تیم مراحل تولید نرم افزار را درک می‌کنند. آن ها می‌دانند چه چیزی قرار است پیش بیاید و چگونه برای اتفاقات بعدی آماده شوند.

۴-موانع بین تیم ها را می شکند .مراحل توسعه نرم افزار ارتباط متقابل بین تیم های توسعه را از طریق پیوند دادن آنها با پروسه ای که نیاز به همکاری دارد برقرار می کند.

۵–دید بهتری از مسئولیت ها و وظایف پیدا می کنند. هر مرحله از چرخه توسعه نرم افزار هدف مشخصی دارد. در نتیجه تیم ها از مسئولیت های بخش های کاری خود آگاه هستند.

مراحل توسعه نرم افزار

در این قسمت با مراحل توسعه نرم افزار و فرآیندهای توسعه نرم افزار آشنا می شوید.

۱- برنامه ریزی و جمع آوری نیازمندی های اولیه برای پروژه تولید نرم افزار

گام اول از مراحل توسعه نرم افزار شامل تعیین نیاز است. به این معنی که درخواست کننده نرم افزار، به راحتی مشکلی را که به آن برخورد کرده برای توسعه دهندگان شرح می‌دهد.

در مرحله اول از مراحل توسعه نرم افزار، اسناد مورد نیاز از داده‌های بدست آمده از مشتری(گاهی اوقات از کاربران) گردآوری و تولید می‌شوند. در نتیجه تیم توسعه متوجه هدفی که می خواهند به آن برسند، می شود.

۲-طراحی

مرحله طراحی نیاز به توسعه معماری، نمونه های اولیه و طراحی تجربه کاربر (UX) دارد. معماری نرم افزار در واقع طرح اولیه تیم توسعه است و شامل فرآیند تولید یک زنجیره متوالی از مدل های یک برنامه نرم افزاری است. این فرآیند برای کنترل کیفیت، وضوح و دسترسی مورد استفاده قرار می گیرد.

در مراحل توسعه نرم افزار این مرحله اهمیت زیادی دارد. در این مرحله تیم توسعه نرم افزار یک نمونه اولیه یا نسخه اولیه از یک رابط کاربری/تجربه کاربر نرم افزار (UI/UX) را ایجاد می کند که این نمونه باید ظاهر و ترتیب عناصر طراحی نرم افزار را به سادگی و وضوح نشان دهد. این گام از این جهت دارای اهمیت است که به مخاطبان این امکان را می دهد که یک تصور بصری از نرم افزار داشته باشند.

مرحله کدنویسی به عنوان گام بعدی در مرحله طراحی، است که در آن تیم توسعه نرم افزار ایده های خود را عملی می کنند.

۳- تست و تضمین کیفیت

این مرحله، در رابطه با اعتبار سنجی کدهای نوشته شده می باشد. در این مرحله تیم تست و تضمین کیفیت تلاش دارد که نقص ها و سایر ناهمگونی های برنامه را رفع کند. در این زمان تیم تست برای دادن گزارش اشکالات به تیم توسعه و تایید آنها تلاش می کند.

٤- توسعه نرم افزار

پس از خلق نرم افزار، تست، اصلاح، تست مجدد و تایید شدن، نرم افزار وارد مرحله توسعه می‌گردد.

مرحله توسعه و استقرار نرم افزار به عملیات معرفی یک محصول عملیاتی به بازار می پردازد.

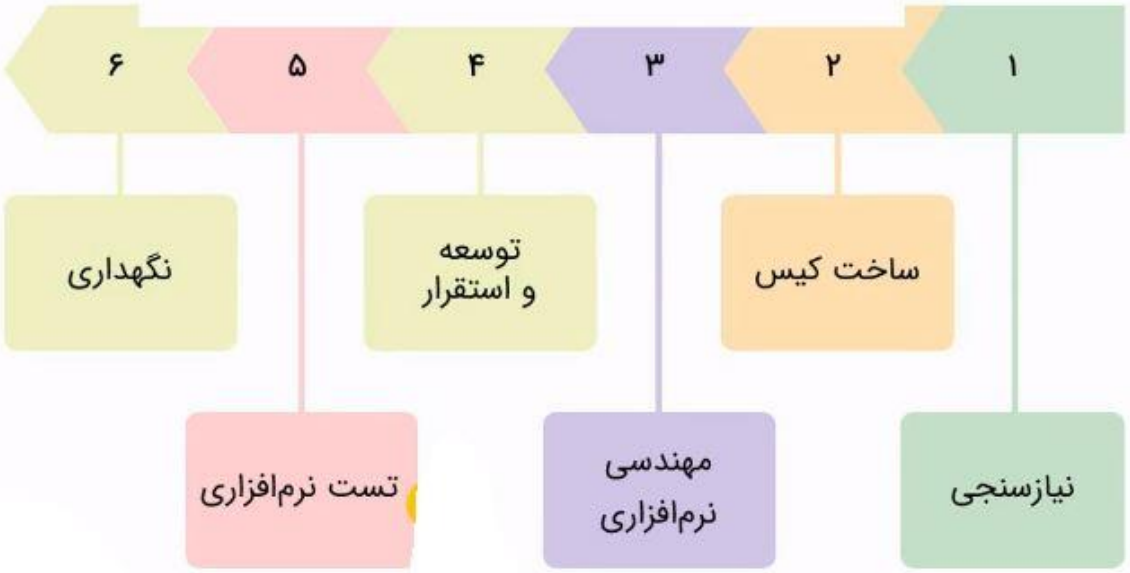
٥-انتشار نرم افزار

یکی از گام های نهایی و کلیدی از مراحل توسعه نرم افزار، مرحله انتشار نرم افزار و ارائه سیستم تولید شده به مصرف کنندگان است. این مرحله از اهمیت زیادی برخوردار است زیرا اگر کاربران در زمان استفاده از نرم افزار، با اشکالی در برنامه تولید شده روبرو شوند، تیم توسعه و برنامه نویسان حرفه ای و متخصص باید برای بررسی مجدد و اصلاح اقدام کنند.

٦-عیب یابی مجدد

مجدد نرم افزار را عیب یابی نموده واز مرحله اول شروع می نمائیم.

فعالیت‌های مورد نیاز برای توسعه نرم‌افزاری



نیازسنجی

پس از طرح مساله توسط کارفرما مرحله نیازسنجی شروع می‌شود.

در این مرحله باید جزئیات موارد مورد نیاز برای ساخت نرم‌افزاری کاربر را جمع‌آوری کرد.

در این مرحله باید تعداد پرسش آماده کنید تا بتوانید راحت‌تر با کارفرما در ارتباط باشید.

با طرح پرسش باید از اهداف نقاط ضعف نقاط قوت و ذی‌نفعان سیستم مطلع شوید.

سوال‌های نیازسنجی

سوال‌های مورد نیاز برای این بخش باید شامل:

- نرم‌افزاری به چه اشخاصی خدمت‌رسانی می‌کند؟
- کاربران شما چه اشخاصی می‌باشد؟
- ماهیت کار شما چه می‌باشد؟

ساخت کیس

پس از جمع‌آوری نیازمندی‌ها باید کیس‌های مرتبط با توسعه را بنویسید.

در این مرحله باید جزئیات موارد مورد نیاز برای ساخت نرم‌افزاری کاربر را جمع‌آوری کرد.

در این مرحله باید تعداد پرسش آماده کنید تا بتوانید راحت‌تر با کارفرما در ارتباط باشید.

در نهایت باید هر کدام این موارد به صورت یک فاز دربیایند.

مهندسی نرم‌افزاری

در این مرحله با توجه به کیس‌ها و نیازمندی‌های شناسایی شده باید زبان‌های مورد نیاز برنامه‌نویسی و توسعه و همچنین زیرساخت‌های مورد نیاز انتخاب شود.

زبان‌های برنامه‌نویسی باید بر اساس نوع کارکرد نرم‌افزاری انتخاب شود.

همچنین زیرساخت‌ها باید به طوری انتخاب شود که توانایی پاسخگویی به کاربران را داشته باشد.

توسعه و استقرار

در این مرحله توسعه شروع می‌شود و با توجه به زبان‌ها و زیرساخت‌هایی که در مرحله مهندسی نرم‌افزاری انتخاب شدن اینکار را انجام می‌دهیم.

در این مرحله از نظر کارایی و کاربردپذیری نرم‌افزاری را مورد تست قرار می‌دهیم.

پس از اتمام این مراحل باید نرم‌افزاری تولید شده را در زیرساخت (سرور و یا هاست) استقرار بدهیم.

تست نرم‌افزاری

پس از اتمام پروژه و توسعه نهایی وارد فرایند تست و نگهداری نرم‌افزاری می‌شویم.

در این مرحله از نظر کارایی و کاربردپذیری نرم‌افزاری را مورد تست قرار می‌دهیم.

فرق این تست با تستی که در مرحله توسعه و استقرار می‌باشد در این است که این فرایند پس از تحویل پروژه هم ادامه‌دار می‌باشد.

نگهداری

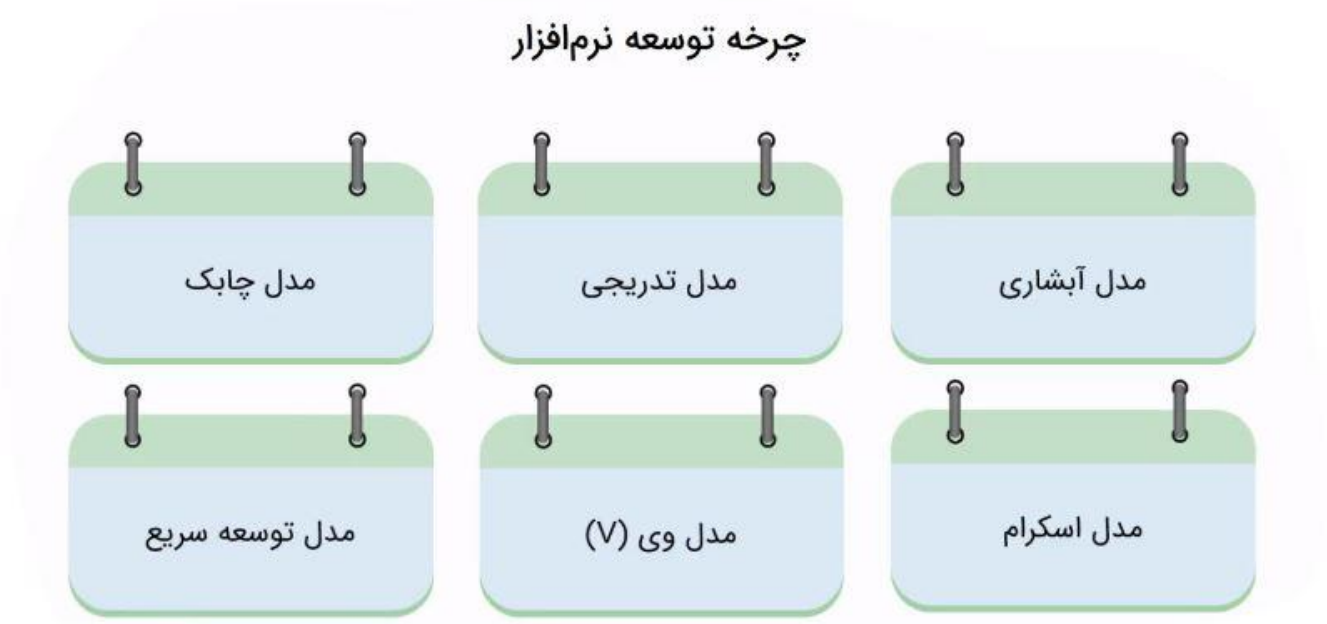
در نهایت پس از طی شدن فرایند مهندسی و تحویل پروژه به مرحله نگهداری می‌رسیم، در این مرحله به توسعه، افزودن فیچر و بالا نگه‌داشتن نرم‌افزاری می‌پردازیم.

این مرحله برای پروژه‌هایی که کارفرمایی غیر از شما دارد انتخابی می‌باشد و ممکن است، مورد نیاز نباشد.

چنانچه کارفرمای نرم‌افزاری، شخص شما باشید باید این مرحله را به صورت دائمی انجام دهید.

این مرحله همراه مرحله تست نرم‌افزاری از مهم‌ترین مراحل پس از اتمام توسعه می‌باشد.

انواع مدل های توسعه نرم افزار:



گیت چیست؟

Gitیک سیستم کنترل نسخه است که برای ردیابی تغییرات در فایل‌های کامپیوتری استفاده می‌شود. به طور کلی برای مدیریت کد منبع در توسعه نرم افزار مورد استفاده قرار می‌گیرد.

بصورت کامل وظایف گیت بصورت زیر است:

Gitبرای ردیابی تغییرات در کد منبع استفاده می‌شود

ابزار کنترل نسخه توزیع شده برای مدیریت کد منبع است.

به چندین توسعه دهنده اجازه می‌دهد تا با هم کار کنند.

این توسعه غیر خطی را از طریق هزاران شاخه موازی خود پشتیبانی می‌کند.

از ویژگی های Git می‌توان به موارد زیر اشاره کرد:

تاریخچه را دنبال می‌کند

رایگان و متن باز

پشتیبانی از توسعه غیر خطی

نسخه پشتیبان تهیه می‌کند

مقیاس پذیر

انشعاب در آن راحت‌تر است

توسعه توزیع شده

جریان عملکردی Git چگونه است؟

گردش کار Git به سه حالت تقسیم می‌شود:

فعالیت بصورت دایرکتوری – فایل‌های موجود در فهرست کاری خود را می‌توانید تغییر دهید.

منطقه مرحله‌بندی (شاخص) – فایل‌ها را مرحله‌بندی کرده و عکس‌های فوری آن‌ها را به منطقه مرحله‌بندی خود اضافه کنید.

دایرکتوری – Git (Repository) یک commit انجام دهید که عکس‌های فوری را به طور دائم در فهرست Gitشما ذخیره می‌کند. هر نسخه موجود را بررسی کنید، تغییراتی را ایجاد کنید، آنها را مرحله‌بندی کنید.

گیت‌هاب (GitHub) چیست؟

احتمالا تا به امروز اسم گیت هاب را شنیده‌اید. گیت‌هاب سرویس میزبانی برای گیت بوده و تحت‌وب ارائه می‌شود. با استفاده از گیت‌هاب، می‌توانید مخازن گیت را برای خود به‌راحتی به‌اشتراک گذاشته و فعالیت‌های تیم خود را به بهترین نحو ممکن مدیریت کنید. به‌عبارت‌دیگر، GitHubمکانی است که در آن، تمامی کارهای مدیران پروژه و برنامه‌نویسان ثبت و ذخیره شده و درنهایت، می‌توان تمامی عملیات را از آن طریق پیگیر شد. در گیت‌هاب می‌توان موارد زیر را انجام داد:

مخازن گیت را می‌توان به‌صورت شخصی و تیمی ذخیره کرد.

فایل‌های خود را از گیت هاب می‌توان دریافت کرد.

GitHubباعث هماهنگی خواهد شد.

ابزارهای متفاوت گیت به افراد کمک می‌کند تا کارهای خود را سریع و بهتر انجام دهند.

چرا از گیت استفاده کنیم؟

دلایل متعددی برای استفاده از Git وجود دارد که با هم بررسی می‌کنیم:

کارایی استاندارد :Git گیت از سیستم‌های کنترل نسخه مانند CVS و SVN و Mercurial قوی‌تر عمل

می‌کند. انجام بروزرسانی‌ها، شاخه‌بندی، سازگاری و مقایسه نسخه‌های قبلی همگی برای مدیریت پروژه شما

در آن امکانپذیر است git .از سیستم توزیع درختی استفاده کرده و برخلاف برخی از نرم‌افزارهای کنترل

نسخه، هنگام ذخیره تاریخچه و نسخه‌های فایل، فقط روی محتوای فایل متمرکز می‌شود.

امنیت خوب :هدف اصلی گیت مدیریت سورس کدهاست و محتویات فایل‌ها و ارتباط بین فایل‌ها و دایرکتوری‌ها، نسخه‌ها، تگ‌ها، Commitها و... در مخزن Git رمزنگاری می‌شوند. این رمزگذاری باعث می‌شود تغییر نسخه‌های قبلی در مخزن غیرممکن باشد و گیت قادر است کدهای شما را بدون هیچ تغییری بازیابی کرده و تحویل دهد. به همین علت همیشه تاریخچه معتبری از کدهای خود رادر اختیار دارید.

منعطف بودن git :بسیار منعطف بوده و از بسیاری دستورها و قابلیت‌ها در پروژه‌های کوچک و بزرگ

پشتیبانی کرده و با بیشتر سیستم عامل‌ها سازگار است.

سرعت خوب Git :از فشرده سازی SHA استفاده می‌کند که باعث بسیار سریع‌تر شدن می‌شود.

ادغام تضادها Git :می‌تواند تداخل‌های ادغام را مدیریت کند، به این معنی که برای کار کردن همزمان چند نفر روی یک فایل یکسان، مشکلی ندارد. این امر دنیای توسعه را به گونه‌ای نشان می‌دهد که با کنترل نسخه متمرکز امکان پذیر نیست. شما به کل پروژه دسترسی دارید و در هر جایی که کار می‌کنید، می‌توانید هر کاری را که نیاز دارید انجام داده و بدانید که تغییرات شما ایمن می‌مانند.

آغاز کار باGit

بسته به سیستم عامل شما، ممکن است قبلاً Git را نصب کرده باشید. اما، شروع کار ما به معنای چیزی بیشتر از داشتن نرم افزار است. برای شروع، دانستن اصول اولیه نحوه عملکرد Git مهم است. می‌توانید کار واقعی را در برنامه‌ای مانند GitHub Desktop یا از طریق GitHub.com انجام دهید) .توجه :در حالی که می‌توانید از طریق GitHub.com با Git تعامل داشته باشید، امکان محدودیت وجود دارد. بسیاری از ابزارهای محلی می‌توانند به شما امکان دسترسی به پرکاربردترین عملکردهای Git را بدهند(.

راه‌های زیادی برای استفاده از Git وجود دارد که لزوما عملکرد آسانی ندارند. اما، گردش کار اساسی Git دارای چند مرحله اصلی است.

ایجاد یک شاخه مشخص

شاخه اصلی معمولاً main نامیده می‌شود. ما می‌خواهیم روی شاخه دیگری کار کنیم، بنابراین می‌توانیم یک درخواست در این زمینه داشته باشیم و با خیال راحت تغییرات را ایجاد کنیم. برای شروع، یک شاخه از mainایجاد کنید. آن را هر طور که می‌خواهید نام گذاری کنید – اما توصیه می‌کنیم شاخه‌ها را براساس

عملکرد یا ویژگی آن، نام گذاری کنید. یک شخص ممکن است چندین شاخه داشته باشد و در یک تیم ممکن است چندین نفر در آن همکاری داشته باشند – شاخه‌ها برای یک هدف هستند نه یک شخص.

ایجاد تغییرات

هنگامی که یک شاخه ایجاد کردید و نشانگر HEAD را با "checking out" به آن شاخه منتقل کردید، آماده شروع کار هستید. با استفاده از ویرایشگر متن یا IDE مورد علاقه خود، تغییرات را در مخزن خود ایجاد کنید .در مرحله بعد، تغییرات خود را ذخیره کنید. شما برای شروع commit آماده‌اید!

برای شروع commit خود، باید به Git اطلاع دهید که چه تغییراتی را می‌خواهید در [git add [file اعمال کنید. هنگامی که تغییرات را ذخیره و مرحله بندی کردید، آماده انجام commit با git commit -m "Descriptive commit message"هستید.

اعمال حالت کنترل از راه دور

تا کنون، اگر یک commit را به صورت محلی انجام داده‌اید، تنها شما هستید که می‌توانید آن را ببینید. برای اینکه دیگران کار شما را ببینند و شروع به همکاری کنند، باید تغییرات خود را با استفاده از git push ذخیره کنید. اگر برای اولین بار از شاخه‌ای که به صورت محلی ایجاد کرده‌اید git push می‌کنید، ممکن است لازم باشد اطلاعات بیشتری به Git بدهید [git push -u origin [branch-name .به گیت می‌گوید که شاخه فعلی را درنظر بگیرید و یک شاخه در کنترل از راه دور ایجاد کنید که با همان نام مطابقت دارد – همچنین،

یک رابطه با آن شاخه ایجاد کنید. به‌طور پیش‌فرض، git pushفقط شاخه‌ای را که در حال حاضر با آن کار می‌کنید در نظر می‌گیرد.

VCS یا Version Control System چیست؟

سیستم کنترل نسخه را می‌توان نرم‌افزاری دانست که به شما کمک می‌کند تا تغییراتی که در طول زمان توسعه روی یک پروژه اعمال می‌کنید را دنبال کنید. در این سیستم زمانی که شما در حال تغییر کدهای‌تان هستید، یک نسخه از کدهای قدیمی‌تان ذخیره خواهد شد. در اصطلاح خود سیستم کنترل نسخه به این کار Snapshot گرفتن از فایل‌ها می‌گویند. سیستم کنترل نسخه اسنپ‌شات گرفته شده را به صورت دائمی در خود نگه‌داری می‌کند، بنابراین شما در هر زمانی می‌توانید به آن دسترسی داشته باشید و آن را فراخوانی کنید.

اما شاید این سوال را بپرسید که قبل از بوجود آمدن سیستم کنترل نسخه این فرایند به چه صورتی انجام می‌شد؟ روش سنتی مدیریت پروژه که هنوز هم استفاده می‌شود نگه‌داری کردن از نسخه‌های مختلف پروژه در پوشه‌های متفاوت است. این کار روش تمیزی برای مدیریت پروژه به حساب نمی‌آید و از طرفی دیگر چندان دقیق نیست. سیستم کنترل نسخه این مشکل را حل می‌کند چرا که می‌تواند تمام نسخه‌های نرم‌افزار شما را به یک صورت خارق‌العاده مدیریت کرده و همچنین دسترسی شما را به این نسخه‌ها حفظ کند.

فواید سیستم کنترل نسخه:

جریان کاری هماهنگ

یکی از مشکلات اصلی روش قدیمی آن است هرکسی در خلال توسعه پروژه از ابزارها و رویکردهای منحصر به فرد خود استفاده می‌کند. این موضوع باعث ایجاد ناسازگاری در پروژه می‌شود. اما سیستم کنترل نسخه توسعه‌دهندگان مختلفی که روی یک پروژه کار می‌کنند را مجبور می‌کند تا از یک جریان کاری هماهنگ استفاده کنند.

سادگی در کار با نسخه‌های مختلف

نسخه‌های مختلف نرم‌افزاری در این سیستم به خوبی و واضحی تعریف می‌شوند. شما برای هر نسخه از نرم‌افزار در سیستم کنترل نسخه یک سری توضیحات خواهید داشت که کمک می‌کند تا تغییرات اعمال شده در این نسخه را مشاهده کنید. مدیریت چندین نسخه در این حالت بسیار پایدارتر و ساده‌تر از مدیریت صدها پوشه است.

قابلیت کدنویسی تیمی

سیستم کنترل نسخه، نسخه‌های نرم‌افزاری مختلفی را که افراد متفاوت در یک تیم نوشته‌اند هماهنگ می‌کند. این کار باعث می‌شود تا از ناسازگاری بین نسخه‌ها و ایجاد برخوردهای نرم‌افزاری پیشگیری شود.

نگه‌داری از تاریخچه تغییرات

سیستم کنترل نسخه تاریخچه‌ای از همه موضوعات مربوط به پروژه را نگه‌داری می‌کند. این تاریخچه می‌تواند شامل آن باشد که چه کسی، چرا و چگونه کدها را در نسخه‌های مختلف تغییر داده است. دانستن این جزئیات به ما کمک می‌کند تا بتوانیم از هر قسمت مربوط به نسخه‌های پروژه سر در بیاوریم.

ویژگی‌های خودکارسازی

ویژگی‌های خودکارسازی که سیستم کنترل نسخه به شما می‌دهد این امکان را دارد تا انجام برخی از کارها در روال سریع‌تر و بهتری صورت بگیرد.

استفاده از Git

اگر تا به حال از یک برنامه‌ی رایانه‌ای یا یک بازی ویدئویی استفاده کرده باشید، متوجه شده‌اید که می‌توانید به نسخه‌ی از پیش ذخیره شده برگردید، اینجاست که نیاز به Git را درک می‌کنید. این کار به سادگی ذخیره‌ی یک اسنپ‌شات از برنامه‌ی شما در زمان است و تغییرات بین کدی که اکنون دارید و آخرین باری که ذخیره کردید را یادداشت می‌کند و دیگر نیازی نیست که هر خط کد از برنامه خودتان را پیگیری کنید. Git یادداشت‌های در حال اجرا را در یک پوشه‌ی پنهان مخصوص ذخیره می‌کند.

اجازه دهید این برنامه‌ی جاوااسکریپت را در نظر بگیریم. این سه خط روی کنسول چاپ می‌شود (خروجی که در مرورگر یا ترمینال خود می‌توانید ببینید):

```
console.log('Hello, this is a git example!')
console.log('And here is another!')
console.log('And yet a third')
```

دستور git init

اگر بخواهم نسخه‌های کارم را ذخیره کنم، می‌توانم از Git استفاده کنم. اول git init را در ترمینال تایپ می‌کنم تا بتوانم از آن استفاده کنم. این کار یک پوشه‌ی git. ایجاد خواهد کرد، جایی که Git فایل‌های خود را ذخیره می‌کند.

دستور git add

. git add تمام فایل‌ها را در برنامه‌ی ما ذخیره خواهد کرد. اگر git init را بعد از ایجاد یک فایل یا هر زمانی که فایل جدیدی ساختید اجرا کردید، شما باید به Git فرمان دهید که با این دستور شروع به ردیابی تغییرات در آن‌ها کند.

دستور git commit

سپس من دستور ”git commit -am "Initial commit را تایپ می‌کنم. git commit دستوری است که یک نسخه از کدهای ما ذخیره می‌کند. نام دیگر دستور am-، (flag) می‌باشد. flag a یعنی ما همه‌ی تغییرات خود را ذخیره خواهیم کرد. flag m اشاره می‌کند که ما پس از آن یک پیام ارائه خواهیم کرد، مثل دستور "Initial commit".

چگونه Git تغییرات را ذخیره می‌کند

اگر ما برنامه‌ی خودمان را تغییر دهیم (مانند تغییر متن خط اول)، ممکن است بخواهیم که یک نسخه‌ی جدید ذخیره کنیم. ما حتی می‌توانیم بین نسخه‌ها تغییر ایجاد کنیم، اگر بخواهیم شاهد چگونگی تغییر برنامه‌ی خودمان در طول زمان باشیم.

```
console.log('Now I have changed the first line.')
console.log('And here is another!')
console.log('And yet a third')
```

دستور git diff

این چیزی است که در حین اجرای این فرمان به نظر می‌رسد. Git تفاوت بین کدی که اکنون دارید و آخرین باری که ذخیره شده بود را به شما نشان می دهد. درک کردن این مسائل کمی دشوار است، اما – حذفی‌ها و + الحاقی‌ها هستند. ما متن "سلام، این یک مثال از Git است!" را حذف کردیم و متن "اکنون خط اول را تغییر دادم" را اضافه کردیم. این‌گونه است که Git آن‌چه را که بین نسخه‌ها تغییر کرده است را پیگیری می‌کند.

```
diff --git a/git.js b/git.js
index eb0f1d1..8dbf769 100644
--- a/git.js
+++ b/git.js
```

@@ -۱,۳ +۱,۳ @@
+console.log('Now I have changed the first line.')
-console.log('Hello, this is a git example!')
console.log('And here is another!')
console.log('And yet a third')

حالا ما شاهد تغییرات commit خواهیم بود، می‌توانیم ادامه دهیم و با این دستور commit جدیدی را ایجاد کنیم: git commit -am 'Update first console log'

این کار تغییراتی که در اولین خط از متن انجام داده‌ام را ذخیره خواهد کرد.

دستور git log

ما می‌توانیم commitهایی که با این دستور انجام داده‌ایم را از نو مرور کنیم. اگر همین الان در برنامه آن را اجرا کنم، این خروجی را بدست خواهم آورد:

commit 67627dd44e84a3106a18a19e94cf9f3723e59b3c (HEAD -> master)
Author: amberwilkie <amber@amberwilkie.com>
Date: Wed Apr 22 16:55:39 2020 -0400
Update first console log

commit 49fe4152f474a9674a83e2b014a08828209d2690
Author: amberwilkie <amber@amberwilkie.com>
Date: Wed Apr 22 16:54:59 2020 -0400
Initial commit

ما پیام‌ها، زمان انجام و شناسه‌ی منحصر به فرد commitهای خود را می‌بینیم و می‌توانیم در آینده از این commitهای مرجع استفاده کنیم.

دستور git checkout

اگر می‌خواستیم به گذشته برگردیم و تغییرات مربوط به کد خودمان را از commit قبلی مشاهده کنیم، ما این کار را با فرمان git checkout 49fe4152f474a9674a83e2b014a08828209d2690 انجام می‌دهیم. Git کد ما را در وضعیت موقتی قرار خواهد داد تا ما بتوانیم ببینیم که کدها در اسنپ‌شات به چه صورتی به نظر می‌رسد.

من شناسه را برای اولین commit خودم کپی کردم. اگر این فرمان را اجرا کنم، برنامه در خط اول، متن "سلام، این یک مثال از git است" را به من نشان می‌دهد.

برای بازگشت به آخرین کد، شما باید فرمان git checkout master را تایپ کنید.

Branchها یا شاخه‌ها

اگر در بالا متوجه شده باشید، ما مجبور شدیم که master را تایپ کنیم تا به وضعیت فعلی کد خود برگردیم. چرا؟ چون master نام پیش‌فرض شاخه‌ی اصلی است، جایی که کدهای ما در آن بروز است. Git برای حفظ کدها به عملیات شاخه زدن یا Branching متکی است. شما می‌توانید master را تنه‌ی درخت در نظر بگیرید. ممکن است کنار گذاشته یا موجب تغییراتی شوید، اما در نهایت به تنه‌ی درخت یا همان master بر خواهید گشت. شما می‌توانید برای ایجاد یک شاخه‌ی جدید، از فرمان git checkout استفاده کنید.

فرمان git checkout -b new-branch را امتحان کنید. زمانی از فلگ b- استفاده می‌کنیم که در حال ایجاد یک شاخه‌ی جدید باشیم و بعد از فلگ، نام شاخه‌ی جدید خود را می‌نویسیم. ما می‌توانیم در این شاخه‌ها commitهای زیادی را ایجاد کنیم و سپس آن‌ها را با فرآیندی به نام merging (ادغام کردن)، به master برگردانیم. در دیاگرام زیر، نقطه‌ها نشانگر commit هستند. دو شاخه از master ایجاد شده است. در توسعه‌ی نرم‌افزار، ما غالبا بر خلاف شاخه‌ی master اصلی، این شاخه ها را "feature" می‌نامیم. شاخه‌ی آبی دوباره با master ادغام شده و شاخه‌ی زرد هنوز هم در حال توسعه است. توجه داشته باشید که حتی اگر شاخه‌ی زرد بعد از شاخه‌ی آبی ایجاد شده بود، فقط تغییرات master در آن شاخه قابل مشاهده خواهد بود. اگر بعضی‌اوقات شاخه‌ی سومی ایجاد کنیم، تغییراتی از شاخه‌ی اصلی و آبی در شاخه‌ی سوم جدید وجود خواهد داشت.

دستور git merge

git merge همه‌ی commitهایی که در آن شاخه انجام داده بودیم را می‌گیرد و آن‌ها را به شاخه‌ی master می‌چسباند و کار شما را ذخیره می‌کند.

چرا از شاخه‌ها استفاده می‌کنیم؟

اگر شما به تنهایی کار می‌کنید، منطقی نیست که کارتان را به چند شاخه تقسیم کنید. چرا همه چیز را فقط در master ذخیره نمی‌کنید؟

تا زمانی که در مورد کار تیمی با سایر توسعه‌دهندگان فکر نکنیم، شاخه سازی کاملا واضح نخواهد بود. به این ترتیب، هر توسعه دهنده می‌تواند شاخه‌ی خود را داشته و تا زمانی که نیاز داشته باشد روی featureهای خود کار کند و در زمان مناسب آن را merge نماید.

Github چیست؟

Github یک پلتفرم رایگان و هاست ابری برای کد است که برای استفاده‌ی شخصی به کار می‌رود. این پلتفرم با Git در کامپیوترهای شما و همکارانتان کار می‌کند و به عنوان Origin، برای هر کسی که روی کد کار می‌کند، خدمت خواهد کرد.

شما و همکارانتان به طور متناوب کدهای خودتان را در GitHub آپلود می‌کنید و GitHub به مرور زمان ابزاری را برای کمک به مدیریت تغییرات در کد فراهم می‌کند.

آپلود کدهایتان در Github

ابتدا باید یک حساب Github ایجاد کنید. شما می‌توانید از این حساب برای کل حرفه‌ی برنامه نویسی خود استفاده کنید و نکته‌ی مهمی که بهتر است رعایت کنید این است که از یک اسم حرفه‌ای یا ترجیحا از اسم واقعی خودتان استفاده کنید.

پس از ورود به سیستم، به دنبال + در بالای صفحه بگردید. روی New Repository کلیک کنید. یک نام برای آن انتخاب کنید، بهتر است هم نام همان پوشه‌ای باشد که قبلا commitهای خود را در آن ذخیره کرده بودید. سپس بر روی "Create Repository" کلیک کنید. اکنون می‌توانید آدرس url مورد نظر خود را که به آن هدایت شده‌اید را کپی کنید و در Origin کدهای خودتان تنظیم کنید.

git remote add origin

حالا ما به codebase دستور می‌دهیم که کدهای ما را در کجای فضای ابری ذخیره کند.

ما فرمان git remote add origin <your-url> را تایپ خواهیم کرد که origin را برای مخزن ما تنظیم می‌کند. حالا می‌توانیم origin خودمان را Push کنیم تا فضای ابری را در Github ذخیره کنیم.

git push

با فرض این‌که هنوز در شاخه‌ی master هستیم (یعنی شاخه‌ی دیگری را بررسی نکرده‌ایم)، اکنون می‌توانیم git push را تایپ کنیم و کد به GitHub می‌رود.

مشاهده‌ی کدهایتان

حالا کد شما در Github زندگی می‌کند. مثال‌ها و توضیحاتی که در مورد مراحل Github زدم را به‌طور خلاصه در این‌جا مشاهده می‌کنید.

شما می‌توانید با مشاهده‌ی وضعیت فعلی کد، روی فایل‌ها و پوشه‌های مخزن کلیک کنید. همچنین می‌توانید نسخه‌های قبلی کد را با کلیک کردن بر روی "X commits" در سمت راست، وسط مشاهده کنید. شما لیستی از commitهای انجام شده در repo را مشاهده خواهید کرد و اگر روی آن‌ها کلیک کنید، می‌توانید فایل‌های پروژه‌ی خود را همان‌طور که در آن زمان وجود داشت مرور کنید.

درخواست‌های Pull

ویژگی‌های زیادی برای Github وجود دارد که مهم‌ترین آن‌ها در همکاری با همکاران، ویژگی درخواست Pull است. یک درخواست Pull، که اغلب به آن PR هم گفته می‌شود، راهی برای مدیریت تغییرات دریافتی در codebase است. برای ایجاد این کار، شما باید یک شاخه‌ی جدید را در کامپیوتر خود بسازید و حداقل یک commit را در آن شاخه ایجاد کنید، سپس git push origin head را تایپ کنید تا آن شاخه به Github ارسال شود. شما می‌توانید نام شاخه‌ی خود را به جای head قرار دهید ولی بهتر است که همه چیز را دقیقا با هم هماهنگ کنید.

حالا وقتی به Github برگردید، باید ببینید که شاخه‌ی شما در دسترس است تا یک PR ایجاد کنید.

اگر روی دکمه‌ی "Compare & pull request" کلیک کنید، می‌توانید تنظیمات بسیاری را برای PR خود تغییر دهید. مهم‌تر از همه انتخاب عنوان و شرح توضیحات آن است. اگر با یک تیم به صورت گروهی کار می‌کنید، می‌توانید همکاران خود را تگ کنید تا از آن‌ها بخواهید که کد شما را بازبینی کنند یا به پروژه‌ها چیزی اضافه کنند و یا بسیاری از ویژگی‌های دیگر که احتمالا هنوز هم به آن‌ها اهمیتی نمی‌دهید.

توجه داشته باشید که ما در حال مقایسه‌ی شاخه‌ها هستیم. در اینجا از شما می‌خواهیم که تغییراتی را که در شاخه‌ی (pr-example) داده شده است را به شاخه‌ی master اضافه کنید. اما ما می‌توانیم هر یک از شاخه‌های دیگر را در repo هدف قرار دهیم. پس فعلا متوجه شده‌اید که فقط شاخه‌ی master نیست که می‌توان به آن درخواست Pull داد.

هنگامی که روی دکمه‌ی "Create Pull Request" کلیک می‌کنید، با این صفحه مواجه خواهید شد:

می‌توانید تمام commitهای موجود در این شاخه را مشاهده کنید و همچنین می‌توانید درخواست Pull خودتان را merge کنید. آیا به خاطر دارید وقتی که داشتیم در مورد Git حرف می‌زدیم، چطور می‌توانستیم کد خودمان را merge کنیم؟ ما می‌توانیم همان عمل را با کد میزبانی ابری

در GitHub انجام دهیم. اگر بر روی دکمه‌ی سبز رنگ "request pull Merge" کلیک کنید، تغییرات شما در master ادغام خواهند شد.

دستور git pull

آخرین دستوری که هم اکنون باید بشناسید، pull git است. اگر شما PR خود را به شاخه‌ی master در Github ادغام کنید، اکنون تغییراتی در origin ایجاد می‌شود که هنوز در رایانه‌ی خود ندارید. اگر شما شاخه‌ی master را بررسی کنید و سپس فرمان git pull origin master را اجرا کنید، تغییراتی که چندی پیش ادغام کرده بودید در رایانه‌ی شما خواهند بود.

```
→ git-example git:(master) git pull origin master
From https://github.com/AmberWilkie/git-example
 * branch master -> FETCH_HEAD
Updating 67627dd..38ad2da
Fast-forward
 git.js | 2 +-
 1 file changed, 1 insertion(+), 1 deletion(-)
```

این Fast-forward به Catching up شاخه‌ی master و شاخه‌ی origin ما اشاره دارد. ما این روند را کامل کردیم:

۱. تغییرات در Local

۲. Push کردن به Github و ساخت PR

۳. ادغام کردن PR در داخل master

۴. Pull کردن master در کامپیوتر Local