**Grundlagen Git**

Führen Sie die folgenden beiden Befehle aus, um Git mitzuteilen, wer Sie sind:

|  |  |
| --- | --- |
|  | $ git config --global user.name " King Kong " |
|  | $ git config --global user.email " king-kong@gmail.com " |

Da Sie die Ausgabe von vielen Git-Befehlen im Terminal sehen, ist es am besten, einige hübsche Farben für die Ausgabe zu haben. Führen Sie einfach den folgenden Befehl aus, um die Code-Hervorhebung zu aktivieren:

|  |  |
| --- | --- |
|  | $ git config --global color.ui true |

Mit dem letzten grundlegenden Konfigurationsbefehl können Sie Ihre Git-Konfigurationen anzeigen. Das Ausführen dieses Befehls entspricht dem Anfordern einer Kopie Ihres Vertrags:

|  |  |
| --- | --- |
|  | $ git config --list |

* Im Terminal cmd + I = Style ändern
* Bei <End> q drücken um zu beenden

Um mit unserer Bankmetapher fortzufahren, müssen wir die Bank ausdrücklich auffordern, ein neues Schließfach zu öffnen, um unsere Effekte zu speichern (lesen Sie: Code). Angenommen, Sie haben bereits ein leeres Verzeichnis für Ihr Projekt erstellt, müssen Sie Git explizit bitten, in diesem Verzeichnis ein Schließfach - ein Repository - zu erstellen:

$ git intit

Der Befehl "init" steht für initialize. Sobald Sie "git init" ausführen, initialisiert Git ein verstecktes Verzeichnis namens ".git" im Stammverzeichnis des Projekts. Und Sie erhalten eine Bestätigung, dass Ihr Schließfach bereit ist! Was kommt als nächstes? Vielleicht möchten Sie den Status Ihrer Box wissen: Speichert sie noch etwas? Um den Git-Status zu ermitteln, müssen Sie Folgendes ausführen:

|  |  |
| --- | --- |
|  | $ git Status |

Du wirst den Befehl "git status" ziemlich oft ausführen. Es ist dasselbe wie wenn Sie einen Bankadministrator anrufen, um zu überprüfen, ob Ihre Sachen angekommen sind oder ob etwas in einen anderen Tresor gebracht wurde.

Im Moment gibt Git die folgende Ausgabe aus:

|  |  |
| --- | --- |
|  | $ git Status |
|  | Auf Zweigmeister |
|  | Erstes Commit |
|  | nichts festzuschreiben (Dateien erstellen / kopieren und mit " git add " verfolgen) |

### Git Branches

Sie können Verzweigungen in Git als Pfade betrachten. Stellen Sie sich vor, Sie erkunden ein neues Gebiet und markieren den Hauptweg zum Wasser mit Stangen von jeweils 10 bis 15 Metern. Dieser Hauptpfad ist wie der Hauptzweig, und die Pole sind wie Commits. Wir werden im letzten Abschnitt des Artikels mehr über Branchen sprechen. Im Moment ist es ausreichend zu wissen, dass Git einen Basiszweig namens Masterzweig hat.

### Git Commits

Ein Commit für ein Repository ist eine Momentaufnahme des aktuellen Status des Stammverzeichnisses des Projekts. Da diese Erklärung eigentlich nichts aussagt, müssen wir das zugrunde liegende Konzept abgrenzen.

Nehmen wir an, Sie arbeiten mit ein paar Papieren. Sie haben zehn Texte über Tiere auf separate Blätter geschrieben und möchten notieren, um welche Texte es sich handelt und wann Sie sie geschrieben haben. Nehmen Sie ein weiteres Blatt Papier heraus, nennen Sie es ein "Commit" und schreiben Sie auf dieses Commit-Papier: "Ich habe Text Nr. 1 geschrieben. Es geht um Vögel. Ich habe Text Nr. 2 geschrieben. Es geht um Hunde ..." Dann erstellen Sie eine Kopie von jedem Text. Das Letzte, was Sie tun, ist, diese zehn Kopien zu sammeln, das Festschreibepapier darauf zu heften und sie in eine Schublade zu legen.

Am nächsten Tag schreiben Sie die Originaltexte neu, holen die Kopien aus Ihrer Schublade und vergleichen die Texte. Dies ist im Grunde, was Git tut. Sie erstellen Dateien und schreiben Code in sie. Wenn Sie bereit sind, Sie *verpflichten* Ihre Dateien in einem Repository: Sie Kopien von Dateien erstellen und sie in eine Schublade legen (ein Repository). (Unser innerer Nerd möchte festlegen, dass Git keine Kopien von Dateien in das Repository pusht. Git erstellt eine leichte Darstellung der Projektdateien, um die Leistung zu verbessern.)

Jeden Tag schreiben Sie diese Commit-Nachricht und fügen neue Texte hinzu. Git erstellt einen Verlauf Ihrer Commits, sodass Sie bis zum Beginn der Projektentwicklung zurückverfolgen können, um zu sehen, welche Dateien geändert oder hinzugefügt wurden, wer sie hinzugefügt oder geändert hat und wann.

**Datei hinzufügen: (Git add) 🡪 auch für änderungen an Datei hinufügen**

**$** git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

Grundlagen\_Git.docx

nothing added to commit but untracked files present (use "git add" to track)

Beachten Sie die Meldung "Untracked files" mit der Datei

" Grundlagen\_Git.docx ". Git informiert uns bequemerweise darüber, dass wir dem

Projekt eine neue Datei hinzugefügt haben. Für Git ist das aber nicht genug. Wie Git uns sagt, müssen wir " Grundlagen\_Git.docx "(ACHTUNG: Dateiname sollte ohne Leerzeichen sein) verfolgen. Mit anderen Worten, wir müssen " Grundlagen\_Git.docx " zum Staging-Bereich hinzufügen. Im folgenden Abschnitt werden die grundlegenden Git-Befehle für die Arbeit mit dem Staging-Bereich erläutert.

Angenommen, Sie möchten einige Ihrer wertvollen Effekte in ein Schließfach verschieben, wissen jedoch noch nicht, welche Objekte Sie dort ablegen werden. Im Moment sammelst du die Dinge einfach in einem Korb. Sie können Dinge aus dem Warenkorb nehmen, wenn Sie der Meinung sind, dass sie nicht wertvoll genug sind, um sie in einem Schließfach aufzubewahren, und Sie können die gewünschten Dinge in den Warenkorb legen. Bei Git ist dieser Korb der *Bereitstellungsbereich* . Wenn Sie Dateien in den Staging-Bereich in Git verschieben, *sammeln* Sie Dateien *und bereiten sie* für Git vor, bevor Sie sie in das lokale Repository übertragen.

Damit Git Dateien für ein Commit nachverfolgen kann:

Datei in Ortner hinzufügen: „Grundlagen Git.docx“ in Ortner „Git Versuch“

Kopieren 🡪 Im Terminal im Pfad des Repository „git status“ 🡪 git add Grundlagen Git.docx

Angenommen, Sie haben dem Stammverzeichnis drei weitere Dateien hinzugefügt: My\_little\_helper.docx und git\_gitflow.pdf. Jetzt möchten Sie alle zum Staging-Bereich hinzufügen. Anstatt diese Dateien separat hinzuzufügen, können wir sie alle zusammenfassen:

$ git add My\_little\_helper.docx git\_gitflow.pdf

Sie müssen lediglich Dateinamen eingeben, die nach dem Befehl "add" durch Leerzeichen getrennt sind. Wenn Sie "git status" noch einmal ausführen, um zu sehen, was sich geändert hat, gibt Git eine neue Nachricht aus, in der alle von Ihnen hinzugefügten Dateien aufgelistet sind:

**$** git status

On branch master

No commits yet

Changes to be committed:

(use "git rm --cached <file>..." to unstage)

new file: Grundlagen\_Git.docx

new file: My\_little\_helper.docx

new file: git\_gitflow.pdf

Das Hinzufügen mehrerer Dateien zum Staging-Bereich auf einmal ist wesentlich bequemer! Aber warte eine Sekunde. Was ist, wenn das Projekt enorm wächst und Sie mehr als drei Dateien hinzufügen müssen? Wie können wir ein Dutzend Dateien (oder Dutzende von Dateien) auf einmal hinzufügen? Git nimmt die Herausforderung an und bietet die folgende Lösung an:

|  |  |
| --- | --- |
|  | $ git add . |

Es gibt ein Problem mit dem "git add". Befehl. Da wir gerade im Stammverzeichnis arbeiten, "git add". fügt nur Dateien hinzu, die sich im Stammverzeichnis befinden. Das Stammverzeichnis kann jedoch viele *andere Verzeichnisse* mit Dateien enthalten. Wie können wir Dateien aus diesen anderen Verzeichnissen und den Dateien im Stammverzeichnis zum Staging-Bereich hinzufügen? Git bietet den folgenden Befehl an: Datei wird nur aus dem Repository entfernt ist aber im Ortner noch vorhanden

|  |  |
| --- | --- |
|  | $ git add --all |
|  |  |

Die Option "--all" teilt Git mit: "Finde alle neuen und aktualisierten Dateien im gesamten Projekt und füge sie dem Staging-Bereich hinzu." Beachten Sie, dass Sie auch die Option "-A" anstelle von "--all" verwenden können. Dank dieser einfachen Option "-A" oder "--all" wird der Workflow erheblich vereinfacht.

**Daten entfernen:**

Erinnerst du dich, als wir dir sagten, dass du Dinge aus deinem imaginären Korb nehmen kannst? Git kann auch Dinge aus dem Korb nehmen, indem Dateien aus dem Staging-Bereich entfernt werden. Verwenden Sie den folgenden Befehl, um Dateien aus dem Staging-Bereich zu entfernen:

|  |  |
| --- | --- |
|  | $ git rm --cached meine-datei.ts |
| oder | $ git reset meine-datei.ts |

In unserem Beispiel haben wir den Befehl "rm" angegeben, der für remove steht. Die Option "--cached" kennzeichnet Dateien im Staging-Bereich. Zum Schluss übergeben wir eine Datei, die wir unstage stellen möchten. Git wird die folgende Nachricht für uns ausgeben: Datei wird komplett gelöscht

|  |  |
| --- | --- |
|  | $ rm ' my\_file.ts ' |

## **Committing Changes to Git (git commit -m “Nachricht”)**

-m 🡪 Message

Beginnen wir mit einem kurzen Überblick über das Festschreiben im Git-Repository. Inzwischen sollten Sie mindestens eine Datei haben, die von Git verfolgt wird (wir haben drei). Wie bereits erwähnt, befinden sich nachverfolgte Dateien noch nicht im Repository. Wir müssen sie verpflichten: Wir müssen unseren Korb mit Sachen zum Schließfach tragen. Es gibt mehrere nützliche Git-Befehle, die (fast) dasselbe bewirken: Verschieben (Festschreiben) von Dateien aus dem Staging-Bereich (einem imaginären Korb) in das Repository (ein Schließfach).

Das Festschreiben an ein Repository ist nicht schwierig. Führen Sie einfach den folgenden Befehl aus:

**$** git commit -m "2 datein hinzugefügt"

[master (root-commit) cb0409b] 2 datein hinzugefügt

2 files changed, 0 insertions(+), 0 deletions(-)

create mode 100644 Grundlagen\_Git.docx

create mode 100644 git\_gitflow.pdf

Verwenden Sie den Befehl "commit", um ein Commit für ein Repository durchzuführen. Übergeben Sie als nächstes dem Befehl "commit" die Option "-m", die für "message" steht. Zuletzt geben Sie Ihre Commit-Nachricht ein. Wir haben für unser Beispiel "Add three files" geschrieben, aber es wird empfohlen, aussagekräftigere Nachrichten wie "Add admin panel" oder "Update admin panel" zu verfassen. Beachten Sie, dass wir die Vergangenheitsform nicht verwendet haben! In einer Festschreibungsnachricht muss angegeben werden, was Ihre Festschreibung *bewirkt* : Hinzufügen oder Entfernen von Dateien, Aktualisieren von App-Funktionen usw.

Also, was haben wir bisher gemacht? Wir haben im ersten Abschnitt Dateien zu einem Projektverzeichnis hinzugefügt. Dann haben wir Dateien zum Staging-Bereich hinzugefügt und jetzt haben wir sie festgeschrieben. Der grundlegende Git-Flow sieht folgendermaßen aus:

* Erstellen Sie eine neue Datei in einem Stammverzeichnis oder in einem Unterverzeichnis oder aktualisieren Sie eine vorhandene Datei.
* Fügen Sie dem Staging-Bereich Dateien hinzu, indem Sie den Befehl "git add" verwenden und die erforderlichen Optionen übergeben.
* Übertragen Sie Dateien mit dem Befehl "git commit -m <message>" in das lokale Repository.

Das ist genug, um einen Eindruck von Git zu bekommen. Aber lassen Sie uns zum Festschreiben von Dateien zurückkehren. Wir sollten eine großartige Alternative zum Standardbefehl "git commit -m 'Does something'" erwähnen.

Wenn Sie an einem Projekt arbeiten, werden Sie möglicherweise einige Dateien ändern und mehrmals festschreiben. Gits Ablauf ändert sich nicht wirklich, wenn einem neuen Commit geänderte Dateien hinzugefügt werden. Mit anderen Worten, jedes Mal, wenn Sie Änderungen vornehmen, müssen Sie eine geänderte Datei zum Staging-Bereich (git add) hinzufügen und anschließend festschreiben (git commit). Aber dieser Standardfluss ist mühsam. Und warum sollten Sie Git bitten müssen, eine Datei zu verfolgen, die zuvor verfolgt wurde?

Die Frage ist, wie wir geänderte Dateien zum Staging-Bereich hinzufügen und gleichzeitig festschreiben können. Git bietet den folgenden Super-Befehl:

|  |  |
| --- | --- |
|  | $ git commit -a -m " Mach noch einmal was " |
|  |  |

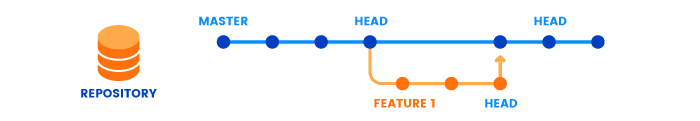
Beachten Sie die Option "-a", die für "add" steht. Git wird auf diesen Befehl wie folgt reagieren: "Ich werde die Dateien einfach sofort festschreiben. Vergessen Sie jedoch nicht, eine Festschreibungsmeldung zu schreiben!" Wie wir sehen können, können wir mit diesem kleinen Trick vermeiden, zwei Befehle auszuführen.

Es wird Zeiten geben, in denen Sie es bereuen, sich für ein Repository entschieden zu haben. Angenommen, Sie haben zehn Dateien geändert, aber nur neun festgeschrieben. Wie können Sie die verbleibende Datei zum letzten Commit hinzufügen? Und wie können Sie eine Datei ändern, wenn Sie sie bereits festgeschrieben haben? Es gibt zwei Auswege. Zuerst können Sie das Festschreiben rückgängig machen:

|  |  |
| --- | --- |
|  | $ git reset --soft HEAD ^ |

Wie Sie sich vielleicht erinnern, ist der Befehl "reset" das Gegenteil des Befehls "add". Dieses Mal weist "reset" Git an, das Festschreiben rückgängig zu machen. Was auf "reset" folgt, ist die Option "--soft". Die Option "--soft" bedeutet, dass das Festschreiben abgebrochen und vor HEAD verschoben wird. Sie können jetzt eine weitere Datei zum Staging-Bereich hinzufügen und festschreiben oder Dateien ändern und festschreiben.

Um zu verstehen, was dieses "HEAD" -Ding darstellt, erinnern Sie sich daran, dass wir in Zweigen arbeiten. Derzeit befinden wir uns in der Hauptniederlassung, und HEAD *verweist*auf diese Hauptniederlassung. Wenn wir später zu einem anderen Zweig wechseln, zeigt HEAD auf diesen anderen Zweig. HEAD ist nur ein Zeiger auf einen Zweig:



Was Sie in der Abbildung sehen, ist, dass jeder Punkt eine separate Festschreibung darstellt und die letzte Festschreibung sich am oberen Rand des Zweigs (HEAD) befindet. Im Befehl "git reset --soft HEAD ^" steht das letzte Zeichen "^" für das letzte Commit. Wir können "git reset --soft HEAD ^" als "Macht das letzte Commit im aktuellen Zweig rückgängig und verschiebt HEAD um ein Commit zurück" lesen.

Anstatt den HEAD zurückzusetzen und die letzte Übergabe rückgängig zu machen, können wir eine Übergabe korrigieren, indem wir die Option "--amend" verwenden, wenn Sie eine Übergabe an ein Repository vornehmen. Fügen Sie einfach die verbleibende Datei zum Staging-Bereich hinzu und bestätigen Sie Folgendes:

|  |  |
| --- | --- |
|  | $ git add file-i-forgot-to-add.html |
|  | $ git commit --amend -m " Füge die verbleibende Datei hinzu " |

Mit der Option "--amend" können Sie *das letzte Commit ändern,* indem Sie eine neue Datei (oder mehrere Dateien) hinzufügen. Mit der Option "--amend" können Sie auch die Nachricht Ihres letzten Commits überschreiben.

Stellen Sie sich diesen Befehl folgendermaßen vor: Sie haben den obersten Papierstapel aus der Schublade genommen und ihn "abgeändert", indem Sie einfach den *Papierstapel entheftet* und ein weiteres Papier, file-i-forgot-to-add.html, hinzugefügt haben und Neuschreiben der Nachricht auf dem "Festschreib" -Papier.

## **Push und Pull zu und von einem Remote-Repository**

In der realen Entwicklung sieht Ihr Workflow folgendermaßen aus:

* Sie arbeiten an einem Feature und übergeben Dateien an einen Zweig (Master oder einen anderen Zweig).
* Sie übertragen Commits an ein Remote-Repository.
* Andere Entwickler übertragen Ihre Commits auf ihre Computer, um die neueste Version des Projekts zu erhalten.

Zuerst müssen Sie ein Remote-Repository erstellen. Wir werden GitHub für diesen Abschnitt verwenden. Sie können ein Konto auf GitHub [erstellen](https://github.com/) und [ein neues Repository](https://github.com/) für Ihr Projekt [erstellen](https://github.com/) . Starten Sie einfach ein Projekt und geben Sie ihm einen Namen. Anschließend müssen Sie den HTTPS-Link zu diesem neuen Repository abrufen. Der Link sieht ungefähr so ​​aus - https://github.com/IhrNutzername/some-small-app.git - wobei IhrNutzername Ihr GitHub-Benutzername und "some-small-app.git" der Name Ihrer App ist .

Jetzt müssen Sie dieses Remote-Repository an Ihr lokales Repository binden:

|  |  |
| --- | --- |
|  | $ git remote add origin https://github.com/Ihr Benutzername/some-small-app.git |

Wir weisen Git an, ein Repository "hinzuzufügen". Die Option "origin" ist der Standardname für den Server, auf dem sich Ihr Remote-Repository befindet. Schließlich gibt es einen Link zu Ihrem Projekt auf GitHub.

Sobald Sie den obigen Befehl ausführen, verbindet Git Ihr lokales und entferntes Repository. Aber was bedeutet diese Verbindung eigentlich? Können Sie bereits online auf Ihren Code zugreifen? Leider noch nicht.

Alles, was Sie bisher getan haben, sind signierte Papiere, damit die Remote Lock Box (GitHub) verschiedene Elemente (Ihren Code) aus Ihrer Home-Schublade (lokales Repository) annehmen kann. Um Ihre Sachen tatsächlich in eine Remote-Schließbox zu kopieren, müssen Sie sie persönlich dorthin tragen. Mit Git sieht das Kopieren Ihres Codes in ein Remote-Repository folgendermaßen aus:

|  |  |
| --- | --- |
|  | $ git push -u origin master |

|  |
| --- |
| $ git push -force origin master |

Um push zu erzwingen

Es ist offensichtlich, dass der Befehl "push" Git anweist, Ihre Dateien in ein Remote-Repository zu verschieben. Was wir auch spezifiziert haben, ist der Server, mit dem unser lokales Repo verbunden ist (Ursprung) und der Zweig, den wir pushen, der Master ist. Es gibt auch diese seltsame "-u" -Option. Das bedeutet, dass wir faul genug sind, nicht jedes Mal einen langen "git push -u origin master" -Befehl auszuführen, wenn wir Code in die Cloud verschieben. Dank "-u" können wir beim nächsten Mal nur "**git push**" ausführen! "

(Wie wir sehen können, gibt es eine SSH-Option, die wir anstelle von HTTPS verwenden können. Wenn Sie Git auf Ihrem Computer für die Arbeit mit SSH eingerichtet haben, müssen Sie die GitHub-Anmeldeinformationen nicht jedes Mal eingeben, wenn Sie den Code an GitHub senden. Sie müssen mit diesem SSH-Link nur einen Remote-Ursprung hinzufügen:)

Nachdem Sie ein Remote-Repository hinzugefügt haben, können Sie die Liste der Repositorys anzeigen, indem Sie den folgenden Befehl ausführen:

|  |  |
| --- | --- |
|  | $ git remote -v |

Bisher haben wir darüber gesprochen, wie Sie Ihre Sachen zu einem Remote-Schließfach bewegen können. Nehmen wir an, Ihre Schublade mit all Ihren Wertsachen ist von zu Hause verschwunden. Jetzt wollen Sie Ihre Sachen wieder aus dem Schließfach holen, als würden Sie sie klonen. Git kann ein gesamtes Projekt aus einem Remote-Repository klonen. Das macht der Befehl "clone":

|  |  |
| --- | --- |
|  | $ git clone git@github.com: YourUsername / your-app.git |
|  |  |

"clone" ist ein einfacher Befehl. Es muss nur ein Link zum GitHub-Projekt übergeben werden. Wir haben eine SSH-Verbindung verwendet, aber Sie können die HTTPS-Verbindung mit demselben Befehl verwenden.

Mit "git clone" wird das gesamte Projekt in ein Verzeichnis auf Ihrem Computer kopiert. Das Verzeichnis wird automatisch erstellt und hat denselben Projektnamen wie das Remote-Repository.

Wenn Ihnen der Name des zu klonenden Repositorys nicht gefällt, geben Sie einfach Ihren bevorzugten Namen an den Befehl weiter:

|  |  |
| --- | --- |
|  | $ git clone git@github.com: YourUsername / your-app.git Dieser Name ist viel besser |

Bisher haben Sie Ihre Änderungen von einem lokalen Repository in ein Remote-Repository verschoben und ein Remote-Repository geklont. Wir haben jedoch nichts über den Befehl "pull" gesagt. Das Übertragen von Änderungen auf GitHub oder BitBucket ist großartig. Wenn andere Entwickler ihre Änderungen in ein Remote-Repository übertragen, möchten Sie ihre Änderungen auf Ihrem Computer sehen. Das heißt, Sie wollen *ziehen* , ihren Code zu Ihrem lokalen Repository. Führen Sie dazu den folgenden Befehl aus:

|  |  |
| --- | --- |
|  | $ git pull |

*Das Klonen* eines Repository unterscheidet sich stark vom *Abrufen* aus einem Repository. Wenn Sie ein Remote-Repository klonen, wird Git:

* Laden Sie das gesamte Projekt in ein bestimmtes Verzeichnis herunter. und
* Erstellen Sie ein Remote-Repository mit dem Namen origin und verweisen Sie es auf die übergebene URL.

Das letzte Element bedeutet einfach, dass Sie nach dem Klonen eines Repositorys nicht "git remote add origin git@github.com: YourUsername / your-app.git" ausführen müssen. Der Befehl "clone" fügt automatisch einen entfernten Ursprung hinzu und Sie können einfach "git push" aus dem Repository ausführen.

Wenn Sie den Befehl "pull" ausführen, wird Git:

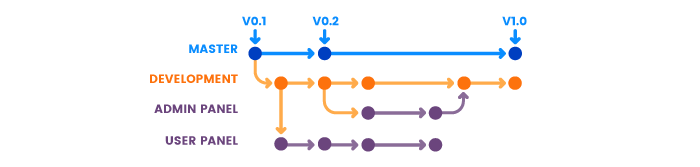
* Ziehen Sie Änderungen in den aktuellen Zweig, die von anderen Entwicklern vorgenommen wurden. und
* Synchronisieren Sie Ihr lokales Repository mit dem Remote-Repository.

Der Befehl "pull" erstellt kein neues Verzeichnis mit dem Projektnamen. Git ruft nur Aktualisierungen ab, um sicherzustellen, dass Ihr lokales Repository auf dem neuesten Stand ist.

## List of Git Commands for Working with Branches

Sie verwenden mehrere Zweige für Ihre Projekte. Zweige sind wohl das größte Merkmal von Git und sie sind sehr hilfreich. Dank Filialen können Sie aktiv an verschiedenen Versionen Ihrer Projekte gleichzeitig arbeiten.

Der Grund, warum wir Äste verwenden, liegt an der Oberfläche. Wenn Sie eine stabile, funktionierende Anwendung haben, möchten Sie diese beim Entwickeln einer neuen Funktion nicht unterbrechen. Daher ist es am besten, zwei Zweige zu haben: einen mit einer stabilen App und einen für die Entwicklung von Funktionen. Wenn Sie ein Feature fertig stellen und es zu funktionieren scheint, ist möglicherweise noch ein Fehler vorhanden. Und Fehler dürfen nicht in einer produktionsbereiten Version auftreten. Daher möchten Sie einen weiteren Zweig zum Testen haben.



Im einfachsten Fall verwenden Sie Zweige, um verschiedene Versionen Ihres Projekts zu speichern: eine stabile App, eine App zum Testen, eine App zur Funktionsentwicklung und so weiter. Was wir beschrieben haben, ist eigentlich nur ein möglicher (aber sicherlich nicht der einzige) Weg, Niederlassungen zu organisieren.

Wie viele Zweige Sie verwenden und wann Sie Zweige erstellen sollten, wird in einem Webentwicklungsteam diskutiert. Vor dem Starten eines Projekts sollten Entwickler entscheiden, wie und wann Zweige erstellt werden sollen, und dann die festgelegten Regeln befolgen, bis das Projekt abgeschlossen ist.

Das Verwalten von Filialen in Git ist einfach. Sehen wir uns zunächst unsere aktuellen Branchen an:

$ git branch

Das war's: Ein Befehl, "branch", fordert Git auf, alle Zweige aufzulisten. In unserer App haben wir nur einen Zweig - Master. Eine in der Entwicklung befindliche Anwendung ist jedoch noch lange nicht vollständig, und wir müssen neue Funktionen entwickeln. Angenommen, wir möchten eine Benutzerprofilfunktion hinzufügen. Um diese Funktion zu erstellen, müssen wir einen neuen Zweig erstellen:

$ git branch manuel

Auch hier ist es ganz einfach: Der Befehl "branch" erstellt einen neuen Zweig mit dem Namen, den wir ihm gegeben haben: "manuel". Können wir sofort Code für unser neues Feature schreiben? Noch nicht!

Beachten Sie das Sternchen links von "master". Dieses Sternchen markiert den aktuellen Zweig, in dem Sie sich befinden. Mit anderen Worten, wenn Sie einen Zweig erstellen und den Code sofort ändern, bearbeiten Sie weiterhin den vorherigen Zweig, nicht den neuen. Nachdem Sie einen neuen Zweig erstellt haben, um ein Feature zu entwickeln, müssen Sie in den neuen Zweig wechseln, *bevor* Sie ein Feature bearbeiten können.

Zum Wechseln von Zweigen in Git wird kein "switch" -Befehl verwendet, wie Sie vielleicht denken. Stattdessen müssen Sie "checkout" verwenden:

|  |  |
| --- | --- |
|  | $ git checkout manuel |
|  | Geschaltete SPRUNG ‚ user-profile ‘ |

**Git Grundlagen 2**

Git benachrichtigt Sie auch darüber, dass Sie zu einem anderen Zweig gewechselt sind: "Zu Zweig 'Benutzerprofil' gewechselt". Lassen Sie uns noch einmal "git branch" ausführen, um zu beweisen, dass:

|  |  |
| --- | --- |
|  | Meister |
|  | \* Benutzerprofil |
|  |  |

**Nach Abschluss der Entwicklung eines Features dieses von diesem Entwicklungszweig auf den Master verschieben? Muss ich kopieren und einfügen? Anonymer Entwickler**

Bevor wir die Fragen beantworten, werfen wir zunächst einen Blick auf den Ablauf beim Hinzufügen neuer Zweige:

1. Erstellen Sie eine neue Verzweigung, um mit "git branch <Verzweigungsname>" eine neue Funktion zu entwickeln.
2. Wechseln Sie mit "git checkout <branch-name>" vom Hauptzweig zum neuen Zweig.
3. Entwickeln Sie die neue Funktion.
4. Du steckst im dritten Schritt fest. Was solltest du als nächstes tun? Die Antwort ist einfach: Sie müssen den Befehl "merge" verwenden. Um einen sekundären Zweig in den Hauptzweig (der ein Master-, Entwicklungs- oder Feature-Zweig sein kann) einzufügen, wechseln Sie zunächst zurück in den Hauptzweig. In unserem Fall sollten wir den Hauptzweig auschecken:

**Git Grundlagen 2**

Git benachrichtigt Sie auch darüber, dass Sie zu einem anderen Zweig gewechselt sind: "Zu Zweig 'Benutzerprofil' gewechselt". Lassen Sie uns noch einmal "git branch" ausführen, um zu beweisen, dass:

|  |  |
| --- | --- |
|  | Meister |
|  | \* Benutzerprofil |
|  |  |

**Nach Abschluss der Entwicklung eines Features dieses von diesem Entwicklungszweig auf den Master verschieben? Muss ich kopieren und einfügen? Anonymer Entwickler**

Bevor wir die Fragen beantworten, werfen wir zunächst einen Blick auf den Ablauf beim Hinzufügen neuer Zweige:

1. Erstellen Sie eine neue Verzweigung, um mit "git branch <Verzweigungsname>" eine neue Funktion zu entwickeln.
2. Wechseln Sie mit "git checkout <branch-name>" vom Hauptzweig zum neuen Zweig.
3. Entwickeln Sie die neue Funktion.
4. Du steckst im dritten Schritt fest. Was solltest du als nächstes tun? Die Antwort ist einfach: Sie müssen den Befehl "merge" verwenden. Um einen sekundären Zweig in den Hauptzweig (der ein Master-, Entwicklungs- oder Feature-Zweig sein kann) einzufügen, wechseln Sie zunächst zurück in den Hauptzweig. In unserem Fall sollten wir den Hauptzweig auschecken:

|  |
| --- |
| $ git Checkout Master |
|  | \* Meister |
|  | Benutzerprofil |

Der aktuelle Zweig wird jetzt in master geändert und der Benutzerprofilzweig kann mit dem Befehl "merge" zusammengeführt werden:

$ git merge user-profile

Denken Sie daran, dass Sie sich in der Hauptniederlassung befinden und eine andere Niederlassung in der Hauptniederlassung zusammenführen - nicht umgekehrt! Da sich die Benutzerprofilfunktion nun im Hauptzweig befindet, wird der Benutzerprofilzweig nicht mehr benötigt. Führen wir also den folgenden Befehl aus:

|  |  |
| --- | --- |
|  | $ git branch -d Benutzerprofil |

Mit der Option "-d" können wir das nun nicht mehr benötigte "Benutzerprofil" löschen. Übrigens, wenn Sie versuchen, den Zweig zu entfernen, in dem Sie sich befinden, lässt Git Sie nicht:

|  |  |
| --- | --- |
|  | Fehler: Der Zweig ' Benutzerprofil ', in dem Sie sich gerade befinden, kann nicht gelöscht werden. |

Anstatt zwei Befehle auszuführen, können Sie nur einen ausführen:

|  |  |
| --- | --- |
|  | $ git checkout -b Admin-Panel |

**Branch auf github über terminal löschen:**

git push origin --delete branchName

git

git push --set-upstream origin Android\_UI\_Tests\_Manuel 🡪 wenn neun Brach vom Terminal auf github erstellen