Crash Intro to Git

This is a short document on how to use the Git version control system.[1] It will teach you the bare minimum you need to know to use it with game development.

What is Git? Git is a "version control system" (VCS): basically, it tries to solve all of the things that can go wrong when multiple people are editing, adding, and deleting files from a big directory all at once (two people trying to edit the same file, nobody knowing who has the newest version of a given file, etc.).

Although Git is a "distributed VCS", you will most likely want to use it in a centralized fashion, as follows:

- You will maintain one central repository with the latest version of your sheets. Most likely, this will be on GitHub or GitLab (two hosting services), but it could also be in an Athena locker or various other locations.
- Each user will have their own "clone" of the repository, which will also have all the sheets. You will write the game by modifying this local copy and "pushing" those changes back to the repository.
- When a user asks for the most up-to-date version of the project, Git will give it to them, being careful not to overwrite changes that they've made on their copy.

We will talk about how to set it up and how to use it. First, make sure you have it installed: run `git --version` at a command prompt. On an Athena machine, or any other machine with git installed, this should return `git version ....` If you don't have git installed and are running Debian or Ubuntu, install it with `apt-get install git`. On other systems, see `https://git-scm.com/downloads` to download it.

1 Creating and populating a git repository

You will only have to do this step once per game.

1.1 Choosing a repo host

You have several options for where to host your repository, with different advantages and disadvantages for each. You will most likely want a host that supports "private" repositories (ones only viewable by your GM team), so that your players don't accidentally spoil themselves. For teams with several members who already use GitHub and with the ability to create private GitHub repos, I suggest using GitHub; otherwise, I suggest GitLab.

GitHub GitHub is probably the biggest commercial Git host, and provides free public repositories to everyone. Unfortunately, private repositories generally require paying money, but if one of your GMs is a *student* (at MIT or elsewhere), they can get free private repositories through the "Student Developer Pack" (`https://education.github.com/pack`). If one of your GMs already pays GitHub for private repos, they can also create another one for free.

GitHub comes with a nice web interface, web-based collaboration tools (of somewhat dubious value for game-writing), and can be integrated with CircleCI[2].

[1] last modified December 2018; based on "Crash Intro to SVN"

[2] CircleCI is a "continuous integration" tool, which can help you discover mistakes you made in your GameTEX before another GM runs into it. However, if nobody on your team is confident with the terminal and git, it's probably more complicated than it's worth. See "Continuous Integration with GameTeX" (`https://adehnert.gitlab.io/TestGame/ci.pdf`) for details.

If you choose to use GitHub, the GM with private repos available should go to `https://github.com/new`, name the repo after your game, mark it as private, and not create a README. After the repo has been created, go to Settings → Collaborators and add all your co-GMs to the list of collaborators.

GitLab GitLab is another commercial Git host with similar features, which provides free public and private repos to everyone. This is my current recommendation for people not already in the GitHub ecosystem or who can't create free private GitHub repos. CircleCI does not appear to work with GitLab, but GitLab has its own built-in CI solution, and "Continuous Integration with GameTeX" describes how to use it as well.

If you choose to use GitLab, create an account at `https://gitlab.com/users/sign_in#register-pane`. You can authenticate with a variety of mechanisms (Google, Twitter, GitHub, Bitbucket) – it appears that it'll use the same username as whatever you log in with, appending a number to make it unique. (Note that it won't prompt you to confirm that you want the username, and will just create it, potentially leaving behind stub accounts if you don't accept the terms of service.) Once you've created an account, you can "Create a group", and then create a "New project". Mark it private and do not initialize the project with a README. Grant all members of your GM team "Owner" access in the "Members" sidebar of the group. (If you're setting permissions at the project level, "Owner" doesn't appear to exist, but you can use "Maintainer" instead.)

Bitbucket Bitbucket is another large commercial Git host, which provides free public repos and free private repos with up to five collaborators each. Bitbucket has many of the same features as GitHub, including CircleCI support, but it's less widely used.

github.mit.edu GitHub also has an "enterprise" version that MIT has purchased. This has much the same web interface as the normal GitHub, has unlimited free private repos, and uses Touchstone for authentication (which is convenient for people with Athena account and usable for those without), but doesn't support CircleCI. If you choose to use github.mit.edu, any GM can go to `https://github.mit.edu/new`, after which the procedure should be similar to the normal GitHub procedure.

Athena You can also host your git repo out of a GM's Athena locker. This loses the web interface of the other systems and requires an Athena account for any collaborators.

You will need an MIT mailing list for your team – create one at `https://listmaker.mit.edu/lc/`. It should be a moira list (*not* mailman) and an AFS group.

One of you should agree to host it in your locker, say in a directory called `gitgamename`. From the command line, you should do several things.

First, create the directory and give your GM team access to it (let's say that your GMs are all on a list called `gamename-gms`. Then, remove everybody else's "list" permissions from it.

```
$ mkdir gitgamename
$ fs sa gitgamename system:gamename-gms write
$ fs sa gitgamename system:anyuser none
```

You can double check the permissions. They should look something like this (assuming that ”joeuser” is the user creating the repo):

```
$ fs la gitgamename
Access list for gitgamename is
Normal rights:
system:gamename-gms rlidwk
system:expunge ld
joeuser rlidwka
```

Then, let's create a fresh, blank repository inside:

```
$ git init --bare gitgamename.git
```

If you look inside the repo, it will have all sorts of mysterious directories.

```
$ ls gitgamename.git/
branches/ config description HEAD hooks/ info/ objects/ refs/
```

Do not manually change anything inside the repository, unless you know how to use git (or are following directions from somebody who does). It will look like garbage anyway. Only use the Git commands we'll talk about in a second.

1.2 Add GameTEX to a repo

Setting up ssh keys *You will only have to do this step once per GM.*

Typically git repos are accessed over ssh. If you're using a GitHub, GitLab, or Bitbucket repo (just about anything but Athena), you'll need an *ssh key*. First, see if you already have one by running `ls $HOME/.ssh/`. If there's a file named `id_something.pub`, you're probably set; if you get an error or see only `config` or `authorized_keys`, you'll need to create a new key. To do so, run `ssh-keygen -t rsa -b 4096 -C "your_email@example.com"`. The default location is fine, and you should use a reasonably-secure password.

Once you have your ssh key, you'll need to upload it to GitHub or whereever else you're hosting your repo – in GitHub's case, go to `https://github.com/settings/keys`; for GitLab, `https://gitlab.com/profile/keys`. You can find your public key by running `cat $HOME/.ssh/id_rsa.pub` (assuming you created a new key above) – *don't* upload the version without the `.pub`.

If you have trouble, GitHub's documentation[3] covers this in more detail – much of it is applicable even if you're using a non-GitHub git host, such as Bitbucket.

Cloning a copy *You will only have to do this step once per GM.*

Each GM needs to clone a local copy of the game's repo.

Decide where your local copy will sit (if using Athena, you probably want it somewhere like `/mit/yourusername/Private/GameName/`, but consult the "Intro to GameTeX" greensheet). Make sure any parent directories exist, change into the parent directory, and then clone the repository. You can do this with the following commands, making sure to replace *`path-to-repo`* with the actual location of the repository (something like `ssh://git@github.com/joeuser/GameName` for GitHub or `/mit/joeuser/Private/gitgamename/gitgamename.git` for Athena).

```
$ mkdir -p /mit/yourusername/Private/
$ cd /mit/yourusername/Private/
$ git clone path-to-repo GameName/
Cloning into 'GameName'...  warning:  You appear to have cloned an empty repository.  done.
```

[3] `https://help.github.com/articles/connecting-to-github-with-ssh/`

(The warning message will appear until one GM does the next step, “Initializing the repo”. However, it’s harmless, and it’s fine for all the GMs to do this step before any does the next one.)

After all is said and done, you will now have a working local copy of the repository. Each GM should do this, including the one who created the repository. Your checked out copy will have a hidden `.git/` folder that you may notice. Don’t mess with it – that’s just how Git keeps track of some information.

Initializing the repo *You will only need to do this step once per game.*

One (and only one) GM will need to add GameTeX to the repository. Before starting this, you should already have the git repository cloned (from the above step), but it should still be empty.

This blank repository needs a copy of your GameTeX tree in it so that you can start using the repo. To do this, you need to *import* the tree so that the repo can start keeping track of it. Set up a copy of the GameTeX tree in your git repository clone. Then run the following commands. A text editor will pop up. Just enter “Initial commit”, save, and exit the text editor.

```
$ git add .
$ git commit
[ edit the commit message ]
$ git push
Counting objects:  98, done.
Delta compression using up to 4 threads.
Compressing objects:  100% (97/97), done.
Writing objects:  100% (98/98), 173.93 KiB | 0 bytes/s, done.
Total 98 (delta 14), reused 0 (delta 0)
To path-to-repo
[new branch] master -> master
```

Now your repo contains a copy of your GameTeX tree and is ready to be used.

2 Normal git operations

You’ll need to do three main things while working on your game:

1. *Pull* your fellow GMs’ changes from the central repo
2. Work on sheets
3. *Push* your changes to the central repo

You’ll learn more about the first and third steps later in this document; the second isn’t git-specific and is thus out-of-scope.

In general, we recommend that you pull immediately before beginning work on your sheets. When you finish a session working (and the game compiles), before you go do other things it’s a good idea to push your changes. Otherwise, if somebody else edits the same file you did, the second of you to push may need to resolve conflicts, which can be tricky. If you pull just before working and push just after, life will be simpler.

Pulling changes from the central repo To pull changes from the central repo, run `git pull`. If nobody else has made changes, you’ll just see:

```
$ git pull
Already up-to-date.
```

If one of your fellow GMs has made changes, you should see something like:

```
$ git pull
remote: Counting objects: 15, done.
remote: Compressing objects: 100\% (9/9), done.
remote: Total 15 (delta 11), reused 10 (delta 6), pack-reused 0
Unpacking objects: 100\% (15/15), done.
From git://github.com/adehnert/TestGame
   daf4cc0..8cae667  master     -> adehnert/master
Updating daf4cc0..8cae667
Fast-forward
 .gitignore                   |  1 +
 Greensheets/guildcamp-git.tex | 85 +++++++++++++++++++++++++++++++++++++++++++++++++++----
 2 files changed, 79 insertions(+), 7 deletions(-)
```

If one of your fellow GMs has made changes to the same place as you did, git will display ”CONFLICT” and you will need to merge them (or throw away your changes). The *Pro Git* book has a section on handling merge conflicts, or you may be able to ask another GM or a friend who knows Git for help.

Examining changes To see what changes you've made, you can run `git status` (to see a summary of changed files) and `git diff` (to see specific changes).

For example, as I write this, `git status` tells me I have one modified file (this one – `guildcamp-git.tex`) and two untracked files (my editor, `vim`'s, tempfiles):

```
$ git status
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
  (use "git push" to publish your local commits)
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   ../Greensheets/guildcamp-git.tex

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

        ../Greensheets/.guildcamp-git.tex.swp
        ../Lists/.green-LIST.tex.swp

no changes added to commit (use "git add" and/or "git commit -a")
```

Running `git diff` tells me exactly what changes I've made to `guildcamp-git.tex`:

```
$ git diff
diff --git a/Greensheets/guildcamp-git.tex b/Greensheets/guildcamp-git.tex
index 8553d3c..c7ecd74 100644
--- a/Greensheets/guildcamp-git.tex
+++ b/Greensheets/guildcamp-git.tex
@@ -157,6 +157,12 @@ used.

 \section{Normal git operations}

+\paragraph*{Examining changes}
+
+To see what changes you've made, you can run \texttt{git status} [...]
+
+
+
 \paragraph*{Committing and pushing changes}
```

Committing and pushing changes Git is a *distributed* version control system, so it tracks changes locally by default and requires a separate operation to push them to the central repo, where your fellow GMs can see it.

The most common way to commit changes is `git commit -a`, which will commit any files you've changed (ones that show up as ”modified” in `git status`) but not unknown files. To commit new files (for example, a new character sheet), run `git add` *`filename`* with each file added to tell git to track those files, followed by `git commit` to store them. Either way, `git commit` will bring up an editor where you can describe your changes (which may be helpful for other GMs to see what you changed).

Once you've stored your changes locally with `git commit`, you can run `git push` to send them to your central repo. If somebody else has pushed since you last pulled, `git` will give you an error – pull, resolve any conflicts, and then try again.

Additional resources GitHub links to a variety of Git (and GitHub) resources at `https://help.github.com/articles/git-and-github-learning-resources/`, including an interactive git course. The whole *Pro Git* book is available free online at `https://git-scm.com/book/en/v2`. The Git project also has a range of other documentation at `https://git-scm.com/documentation`, including videos and a collection of external links.